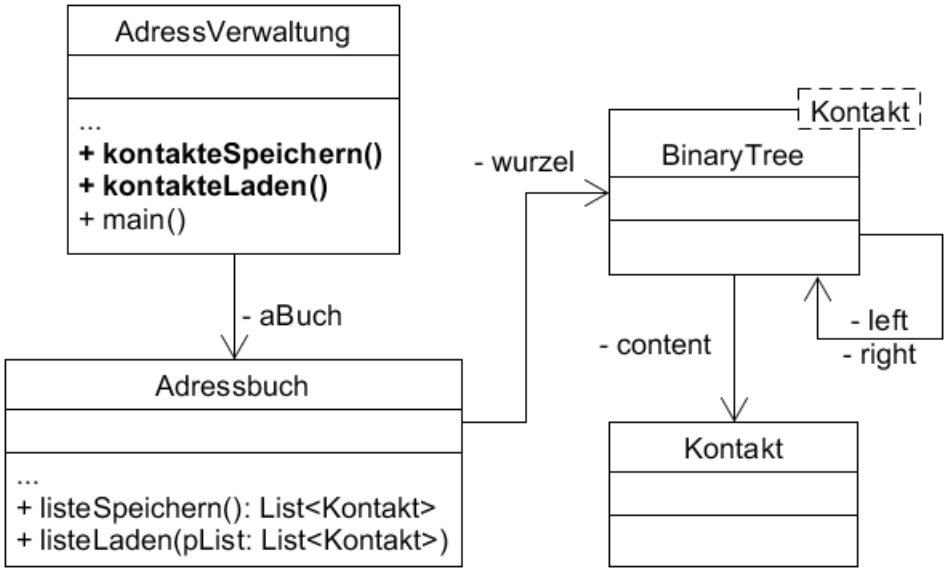
Adressbuch speichern

Für unser Adressbuch wäre es schön, wenn wir die bereits eingegebenen Kontakte in einer Datei speichern könnten. Dann könnte man das Programm schließen und zu einem späteren Zeitpunkt wieder öffnen, und die gespeicherten Kontakte ins Programm laden.

Dazu kannst du die Methoden kontakteSpeichern() und kontakteLaden() der Klasse **AdressVerwaltung** anpassen:



Die Klasse **Adressbuch** enthält bereits die Methoden listeSpeichern() und listeLaden(), die wir für diesen Zweck benutzen können. Diese Klasse braucht nicht weiter bearbeitet zu werden.

Der **Algorithmus** ist nicht schwer zu beschreiben:

1. Wenn das Adressbuch gespeichert werden soll, erzeuge eine lineare Liste der Kontakte aus dem Binärbaum. Dabei wird die Pre-order-Reihenfolge angewendet, damit der Binärbaum später beim Neu-Aufbau wieder genau die gleiche Struktur erhält wie zum Zeitpunkt des Speicherns.

2. Schreibe die in der Liste enthaltenen Kontakte mit allen zugehörigen Daten in eine Textdatei.

3. Später, wenn das Adressbuch aus der Datei geladen werden soll, erzeuge eine neue (leere) Liste. Lies die Daten der einzelnen Kontakte aus der Textdatei, erzeuge damit jeweils ein neues Kontakt-Objekt und füge es am Ende der Liste ein.

4. Wenn alle Kontakte eingelesen und zur Liste hinzugefügt wurden, erzeuge einen neuen Binärbaum aus dieser Liste.

Neu ist nur der **Umgang mit (Text-)Dateien**, der auf den folgenden Seiten erläutert wird.

Textdateien

Der Umgang mit Textdateien kann sehr komplex sein, weil es viele mögliche Fehlerquellen und Sonderfälle gibt, die in einem (professionellen) Programm beachtet werden müssen – zum Beispiel:

- Texte können auf ganz unterschiedliche Art **codiert** werden, z.B. mit dem Unicode-, Windows- oder Apple-Zeichensatz, die zum Teil je nach Land noch unterschiedlich sind.
- Der Name einer Datei muss bestimmten Regeln folgen.
- Beim Schreiben einer Datei kann es passieren, dass der Datenträger (z.B. Festplatte) voll wird.
- Beim Versuch, eine Datei zu lesen, wird diese evtl. nicht gefunden.

  (usw.)

Java bietet zudem eine Fülle verschiedener Klassen für den Umgang mit Dateien, die jeweils ihre Besonderheiten und Möglichkeiten haben. Wer noch nie mit Dateien gearbeitet hat, findet sich in diesem „Dschungel“ nur schwer zurecht.

Das folgende Beispiel zeigt eine einfache Möglichkeit, bei der viele Sonderfälle ausgeblendet werden. Es funktioniert, wenn man nur mit einem Betriebssystem arbeitet und behandelt nicht alle möglichen Fehlerquellen – für unser einfaches Programm reicht das aber völlig aus.

Beispiel: Schreiben von Strings in eine Textdatei

Hinweis: die Klassen File, FileWriter usw. benötigen den Import von `java.io.*`

```
01  public void schreiben(String dateiname, String[] zeilen)
02  {
03     try
04     {
05        File file = new File(dateiname);
06        FileWriter writer = new FileWriter(file);
07
08        for (int i = 0; i < zeilen.length; i++)
09        {
10           writer.write(zeilen[i] + "\n");
11        }
12        writer.close();
13     }
14     catch (Exception ex)
15     {
16        Console.println("Dateifehler: ex.getMessage()");
17     }
18  }
```

Die Methode erhält den Namen der Datei sowie ein Array von Strings als **Parameter**.
Der Dateiname kann z.B. `"kontakte.txt"` sein – dann muss die Datei im gleichen Ordner liegen wie das Java-Programm.
Man kann auch einen Pfad angeben, z.B. `"./saves/kontakte.txt"`.

In den Zeilen 05 und 06 wird die Datei mit der Hilfsklasse `FileWriter` geöffnet.
Falls die Datei noch nicht existiert, wird sie neu angelegt. Falls es die Datei schon gibt, wird ihr Inhalt komplett gelöscht und von den folgenden write-Befehlen überschrieben.

In der for-Schleife (Zeile 10) werden die einzelnen Strings mit der `write`-Methode in die Datei geschrieben. An jeden String wird das Sonderzeichen `\n` angehängt, also ein Zeilenumbruch.

Die Befehle `try` und `catch` (Zeile 03 bzw. 15) muss man an dieser Stelle verwenden. Sie sollen mögliche Ausnahmefälle abfangen, die bei der Arbeit mit Dateien auftreten können. Java stellt für solche Ausnahmefälle sogenannte „**Exceptions**“ zur Verfügung, die an dieser Stelle nur ganz oberflächlich beschrieben werden können. Der try-catch-Mechanismus bedeutet hier:
„Probiere aus, die Datei zu öffnen und hineinzuschreiben. Falls dabei irgendein Fehler auftritt, breche ab und springe zum catch-Befehl.“
Im catch-Block wird schlicht ein Text auf der Konsole ausgegeben, der den Fehler beschreibt.

Beispiel: Lesen von Strings aus einer Textdatei

```
01  public void dateiAusgeben(String dateiname)
02  {
03     try
04     {
05        File file = new File(dateiname);
06        BufferedReader reader =
07           new BufferedReader(new FileReader(file));
08
09        String zeile = reader.readLine();
10        while (zeile != null)
11        {
12           Console.println(zeile);
13           zeile = reader.readLine();
14        }
15        reader.close();
16     }
17     catch (Exception ex)
18     {
19        Console.println("Dateifehler: ex.getMessage()");
20     }
21  }
```

Zum Lesen wird hier die Klasse `BufferedReader` verwendet, da sie im Gegensatz zum einfacheren `FileReader` die Methode `readLine()` zur Verfügung stellt.
Diese Methode gibt bei jedem Aufruf die nächste Zeile der Datei als String zurück.
Irgendwann erreicht man so das Ende der Datei, und die Methode gibt dann null zurück.
Die Datei wird durch diese Lese-Operationen nicht verändert.

Hinweis: Wenn man einzelne Wörter oder Zahlen aus einer Zeile lesen möchte, empfiehlt es sich, stattdessen die Klasse `Scanner` zu verwenden. Bei Interesse findest du diese in der Online-Dokumentation von Java beschrieben.

Aufgabe

Verwende die ausgeteilte BlueJ-Vorlage (die Musterlösung des Projekts zu den Traversierungs-Algorithmen). Die Klassen Kontakt, BinaryTree und Adressbuch sind bereits fertig implementiert, du brauchst nur die **Klasse AdressVerwaltung** zu erweitern.

a) Implementiere die Methode **kontakteSpeichern()**:
Sie bittet den Benutzer, einen Dateinamen einzutippen.
Die vom Benutzer eingegebene Datei wird geöffnet.
Mithilfe der entsprechenden Methode der Klasse Adressbuch wird aus dem Binärbaum eine Liste erzeugt.
Für jeden Kontakt in der Liste schreibt die Methode Nachname, Vorname, Telefonnummer und Emailadresse in die Datei. Wir nehmen zur Vereinfachung an, dass diese Daten nie Zeilenumbrüche enthalten, für jeden Kontakt werden also genau vier Zeilen geschrieben.
Am Ende wird die Datei geschlossen und der Benutzer über den Erfolg der Methode informiert.

b) Implementiere die Methode **kontakteLaden()**:
Sie bittet den Benutzer, einen Dateinamen einzutippen.
Die vom Benutzer eingegebene Datei wird geöffnet, und es wird eine leere Liste für Kontakt-Objekte erzeugt.
Dann werden je vier Zeilen aus der Datei eingelesen für Nachname, Vorname, Telefonnummer und Emailadresse. Daraus wird ein neues Kontakt-Objekt erzeugt und am Ende der Liste eingefügt. Das wiederholt sich, solange Zeilen aus der Datei gelesen werden können.
Aus der Liste kann dann mithilfe der entsprechenden Methode der Klasse Adressbuch ein neuer Suchbaum von Kontakten erzeugt werden.
Am Ende wird die Datei geschlossen und der Benutzer über den Erfolg der Methode informiert.

Zu Beginn der Methode ist es evtl. sinnvoll, wenn der Benutzer die Funktion bestätigen muss, denn falls das Adressbuch schon Kontakte enthält, werden diese während des Ladens gelöscht.