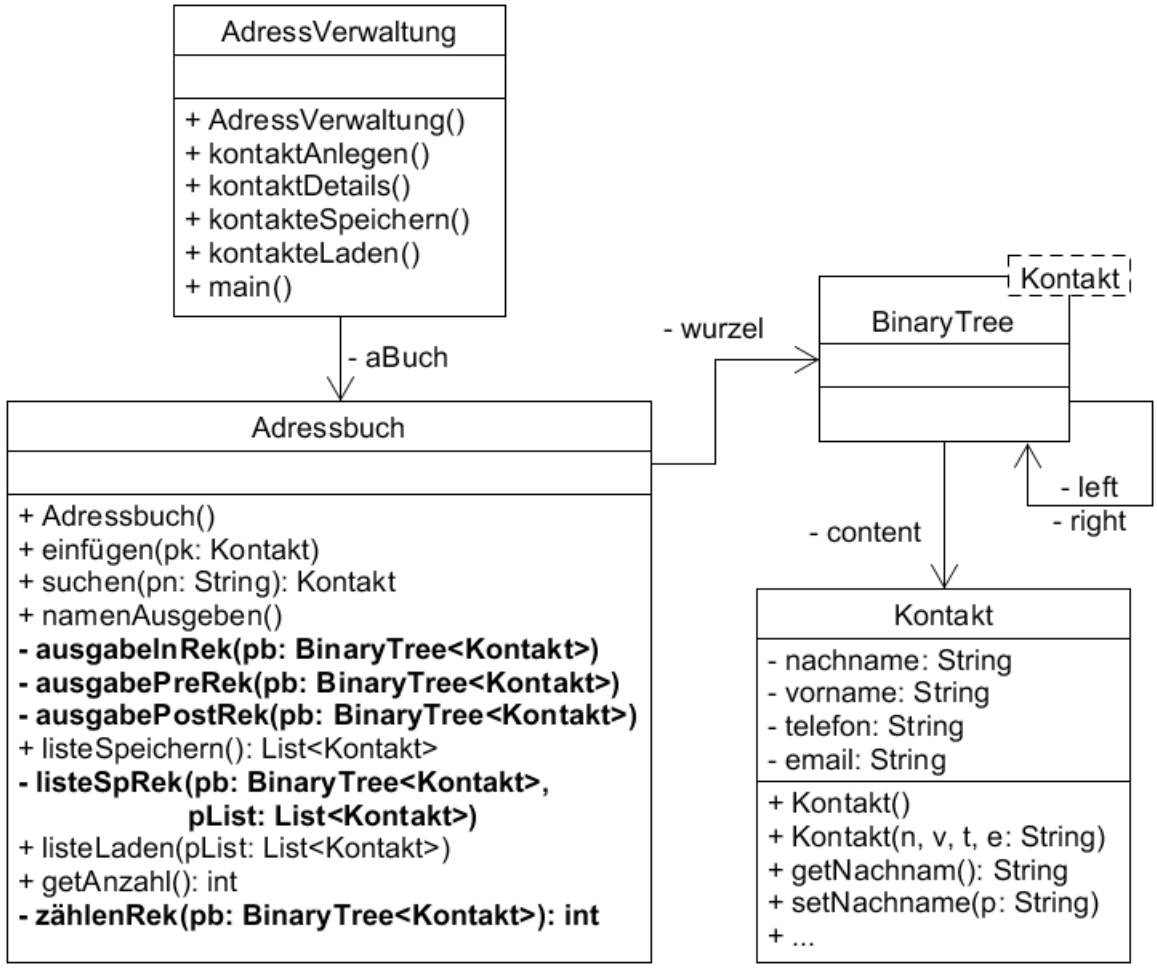
Wir erweitern das Adressbuch um Funktionen, für die wir **Traversierungs-Algorithmen** brauchen: Ausgabe aller Namen auf der Konsole, Zählen der Kontakte und Speichern in einer linearen Liste:



Die Traversierungs-Algorithmen lassen sich elegant und einfach mit Rekursion umsetzen.
Die rekursiven Methoden sind im Klassendiagramm fett gedruckt.

Rekursive Methoden benötigen jeweils einen Parameter für den aktuellen Knoten des Suchbaums. Auf den Suchbaum sollte jedoch außerhalb der Klasse Adressbuch kein Zugriff möglich sein. Daher werden alle rekursiven Methoden mit dem Zugriffsmodifikator private deklariert.

Um die Funktionen für andere Klassen – z.B. hier für die Hauptklasse – zur Verfügung zu stellen, gibt es jeweils eine zweite, öffentliche Methode, die die Funktion quasi „verpackt“. Man nennt diese Methoden auch „Wrapper-Methoden“.

Wrapper-Methode: Beispiel

Eine Wrapper-Methode ruft die rekursive Methode mit dem Wurzelknoten auf. Sie benötigt selbst keinen Parameter und kann daher gefahrlos für andere Klassen zur Verfügung gestellt werden.

```
public class Adressbuch
{
   // Der Binärbaum ist nur innerhalb dieser Klasse „sichtbar"
   private BinaryTree<Adresse> wurzel;

   /*
    * Berechnet die Anzahl der Knoten des Binärbaums „knoten"
    * Kann nur innerhalb der Klasse Adressbuch aufgerufen werden
    */
   private int zählenRek(BinaryTree<Adresse> knoten)
   {
      ... (ruft sich selbst rekursiv auf) ...
   }

   /*
    * Wrapper für die rekursive Methode
    * Startet die Rekursion mit dem Wurzelknoten
    * Kann von anderen Klassen aufgerufen werden
    */
   public int getAnzahl()
   {
      return zähleRek(wurzel);
   }
}
```

Aufgabe 1

Implementiere die Traversierungs-Algorithmen mithilfe der bereitgestellten BlueJ-Vorlage.
Du brauchst nur die Methoden der **Klasse Adressbuch** zu implementieren.

a) Für die Ausgabe aller Namen der Liste bietet die Klasse Adressbuch drei rekursive Methoden, die die Namen in der Preorder-, Inorder bzw. Postorder-Reihenfolge ausgeben.
Implementiere diese Methoden:

```
void ausgabePreRek(BinaryTree<Kontakt> pb)
void ausgabeInRek(BinaryTree<Kontakt> pb)
void ausgabePostRek(BinaryTree<Kontakt> pb)
```

Anschließend implementiere die Wrapper-Methode `void namenAusgeben()`.
Diese sollte die Namen in Inorder-Reihenfolge ausgeben – **warum?**
Probiere aber auch die beiden anderen Reihenfolgen aus.

Die Hauptklasse ist bereits fertig. Der erste Menüpunkt füllt das Adressbuch mit Beispieldaten.
Du kannst deine Methoden also schon testen, wenn du die main-Methode ausführst.

b) Implementiere die rekursive Methode

```
void listeSpRek(pb: BinaryTree<Kontakt>, pList: List<Kontakt>)
```

Beim Start der Rekursion wird der Wurzelknoten und eine leere Liste als Parameter übergeben. Sie fügt alle Kontakte des Binärbaums in Preorder-Reihenfolge in die Liste ein
– **warum Preorder?**

Implementiere anschließend die Wrapper-Methode `List<Kontakt> listeSpeichern()`. Sie erzeugt ein neues List-Objekt und startet die die Rekursion. Dann gibt sie die gefüllte Liste zurück.

Du kannst deine Methode testen, indem du die main-Methode ausführst, das Adressbuch mit Beispieldaten füllst und anschließend die Funktion „In Liste Speichern“ wählst. Die Liste wird auf der Konsole ausgegeben. So kannst du prüfen, ob sie tatsächlich in Preorder-Reihenfolge gefüllt wurde.

c) Implementiere die Methode `void listeLaden(pList: List<Kontakt>)`

Sie erhält eine **gefüllte** Liste als Parameter. Die Methode fügt alle Kontakte der Liste der Reihe nach in den Suchbaum ein. Sie braucht dazu keine Rekursion, sondern kann einfach für jeden Kontakt in der Liste die Methode einfügen() aufrufen.

d) Implementiere die rekursive Methode `int zählenRek(pb: BinaryTree<Kontakt>)` Sie bestimmt die Anzahl der Kontakte im Adressbuch.

Implementiere dann die Wrapper-Methode `int getAnzahl()`, die rekursive Methode startet und die Anzahl der Kontakte zurückgibt.

Aufgabe 2

Für die folgenden Aufgaben gib jeweils einen rekursiven Algorithmus an (in Umgangsprache oder in Java). Du brauchst die Methoden jedoch nicht in deinen BlueJ-Projekt zu implementieren.

a) Bestimme die Anzahl der Blätter (also der Knoten ohne Nachfolger) des Binärbaums.
Beachte, dass in der Implementierung des Schulministeriums jedes Blatt zwei leere Knoten als Nachfolger hat.

b) Bestimme die maximale Tiefe des Binärbaums.