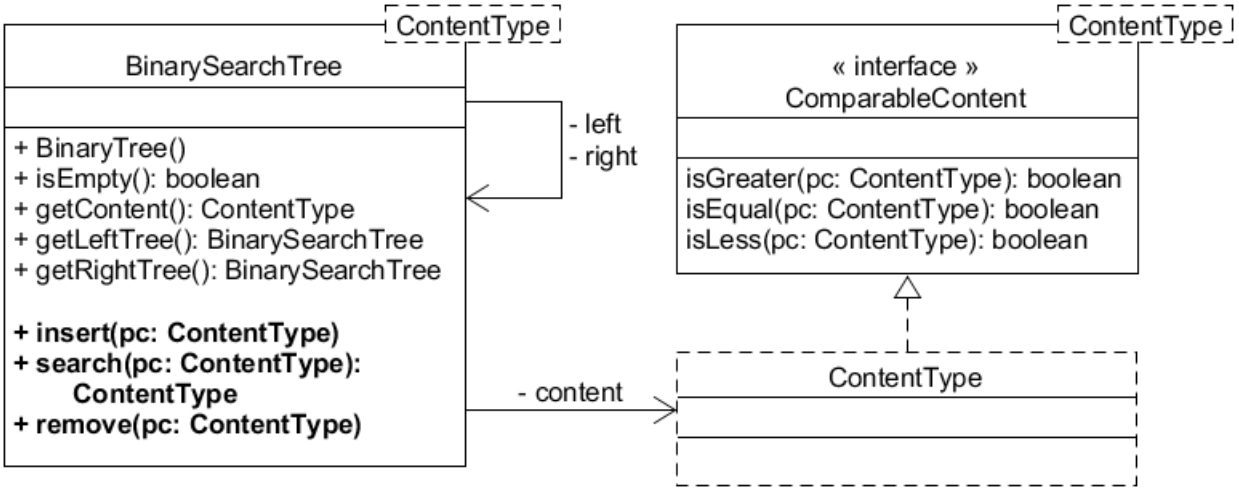
Für das Zentralabitur stellt das Schulministerium NRW eine Klasse zur Verfügung, die einen binären Suchbaum implementiert: die Klasse BinarySearchTree.



Ähnlich wie die Klasse BinaryTree ist BinarySearchTree generisch, d.h. man kann Objekte beliebiger Klassen als Inhaltsobjekte definieren, jedoch mit einer Einschränkung:

Da für das sortierte Einfügen in den Suchbaum eine **Vergleichsoperation** notwendig ist, muss die Klasse der Inhaltsobjekte das **Interface** „ComparableContent“ implementieren.
Dieses Interface gibt vor, dass die Klasse der Inhaltsobjekte die drei Vergleichsmethoden isGreater(), isEqual() und isLess() implementieren muss.

Wenn man für eine Inhalts-Klasse diese drei Methoden implementiert, kann man deren Objekte ganz leicht in den Suchbaum einfügen, im Suchbaum nach Objekten suchen oder diese löschen. Dazu stellt BinarySearchTree die Methoden insert(), search() und remove() zur Verfügung. Anders als bei BinaryTree sind die rekursiven Algorithmen hier bereits implementiert.

Methoden der Klasse BinarySearchTree

Die Methoden isEmpty(), getLeftTree(), getRightTree() und getContent() funktionieren genau wie bei der Klasse BinaryTree. Mit diesen Methoden könnte man den Suchbaum mit den bekannten Algorithmen bearbeiten. Das wäre jedoch wenig sinnvoll, da die Klasse BinarySearchTree alle benötigten Methoden bereits implementiert. Außerdem fehlen die Methoden setContent(), setLeftTree() usw., mit denen man die Knoten des Baums direkt verändern könnte.

Stattdessen nutzt man die folgenden Methoden:

`void insert(ContentType pContent)`
Fügt das Objekt pContent an der richtigen Stelle im binären Suchbaum ein.
Sonderfall: Wenn bereits ein Objekt im binären Suchbaum enthalten ist, das – entsprechend der Vergleichsmethode isEqual() – gleichgroß ist wie pContent, wird das Objekt nicht eingefügt.
(Falls pContent null ist, bleibt der Baum ebenfalls unverändert.)

`ContentType search(ContentType pContent)`
Falls ein Objekt im binären Suchbaum enthalten ist, das „gleichgroß“ ist wie pContent, also isEqual() true zurückgibt, gibt die Methode eine Referenz auf dieses Objekt zurück.
Sonst wird null zurückgegeben. Im Sonderfall pContent = null wird ebenfalls null zurückgegeben.

Wenn der Suchbaum zum Beispiel Kontakte enthält, und der Baum nach den Nachnamen dieser Kontakte sortiert ist, müsste man anhand eines Nachnamens nach einem Kontakt suchen.
Dazu erzeugt man ein temporäres Kontakt-Objekt mit dem gesuchten Nachnamen (die restlichen Attribute können leer bleiben), und setzt dieses temporäre Objekt als Parameter in search() ein.
search() gibt das gesuchte Objekt zurück, das dann die Werte für alle Attribute enthält.

Das bedeutet allerdings, dass die Methode isEqual() nur die Gleichheit bezüglich des Sortierkriteriums prüft – denn völlig gleich sind pContent und das gesuchte Objekt ja nicht.

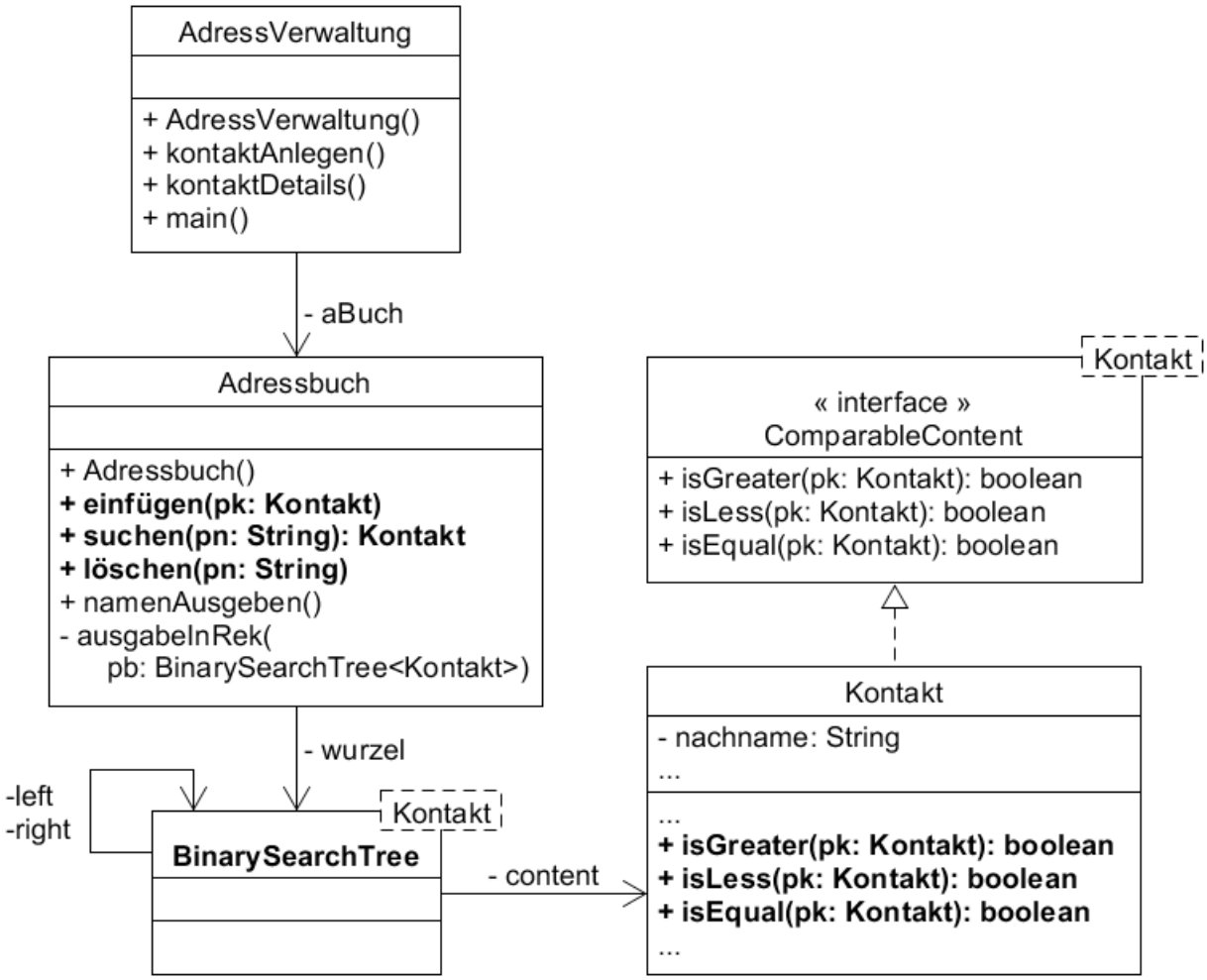
`void remove(ContentType pContent)`
Falls ein Objekt im binären Suchbaum enthalten ist, das „gleichgroß“ ist wie pContent, wird dieses Objekt aus dem binären Suchbaum entfernt.
Sonst, oder falls pContent null ist, bleibt der Baum unverändert.

Aufgabe

Statt das Einfügen, Suchen und Löschen mit der Klasse BinaryTree selbst zu implementieren, können wir nun auch einfach die Klasse BinarySearchTree für das Adressbuch verwenden. Insbesondere für das Löschen eines Kontaktes wird uns dadurch eine Menge Arbeit erspart, denn der Algorithmus ist nicht einfach zu implementieren.

Im Modell wird lediglich die Klasse BinaryTree durch BinarySearchTree ersetzt.

Dazu muss allerdings die Klasse Kontakt das Interface ComparableContent implementieren (den entsprechenden Quellcode findest du in der Vorlage).



a) Für die Klasse **Kontakt** implementiere die Vergleichsmethoden

```
boolean isLess   (Kontakt pk)
boolean isGreater(Kontakt pk)
boolean isEqual  (Kontakt pk)
```

Sie geben true zurück, wenn der Nachname des Kontakt-Objekts alphabetisch vor / nach / auf gleicher Höhe liegt wie der Nachname des als Parameter pk übergebenen Objekts.

b) Für die Klasse **Adressbuch** implementiere die Methoden

i. `void einfügen(Kontakt pk)`
Fügt den Kontakt pk in den Suchbaum ein

ii. `Kontakt suchen(String pn)`
Erzeugt ein neues Kontakt-Objekt mit dem Nachnamen pn, um es dann für die Suchfunktion der Klasse BinarySearchTree zu verwenden.
Falls der gesuchte Kontakt vorhanden ist, wird er zurückgegeben, sonst null.

iii. `void löschen(String pn)`
Wie in der Methode suchen() wird ein Kontakt-Objekt mit dem Nachnamen pn erzeugt. Damit kann der entsprechende Kontakt aus dem Suchbaum mit der Löschfunktion der Klasse BinarySearchTree gelöscht werden.

iv. void ausgabeInRek(pb: BinarySearchTree<Kontakt>) Die rekursive Methode zur Ausgabe der Namen auf der Konsole in In-order-Reihenfolge funktioniert mit dem BinarySearchTree genauso wie mit dem einfachen BinaryTree.
Schau also ggf. in deine Lösung der vorigen Aufgaben oder in die Musterlösung.