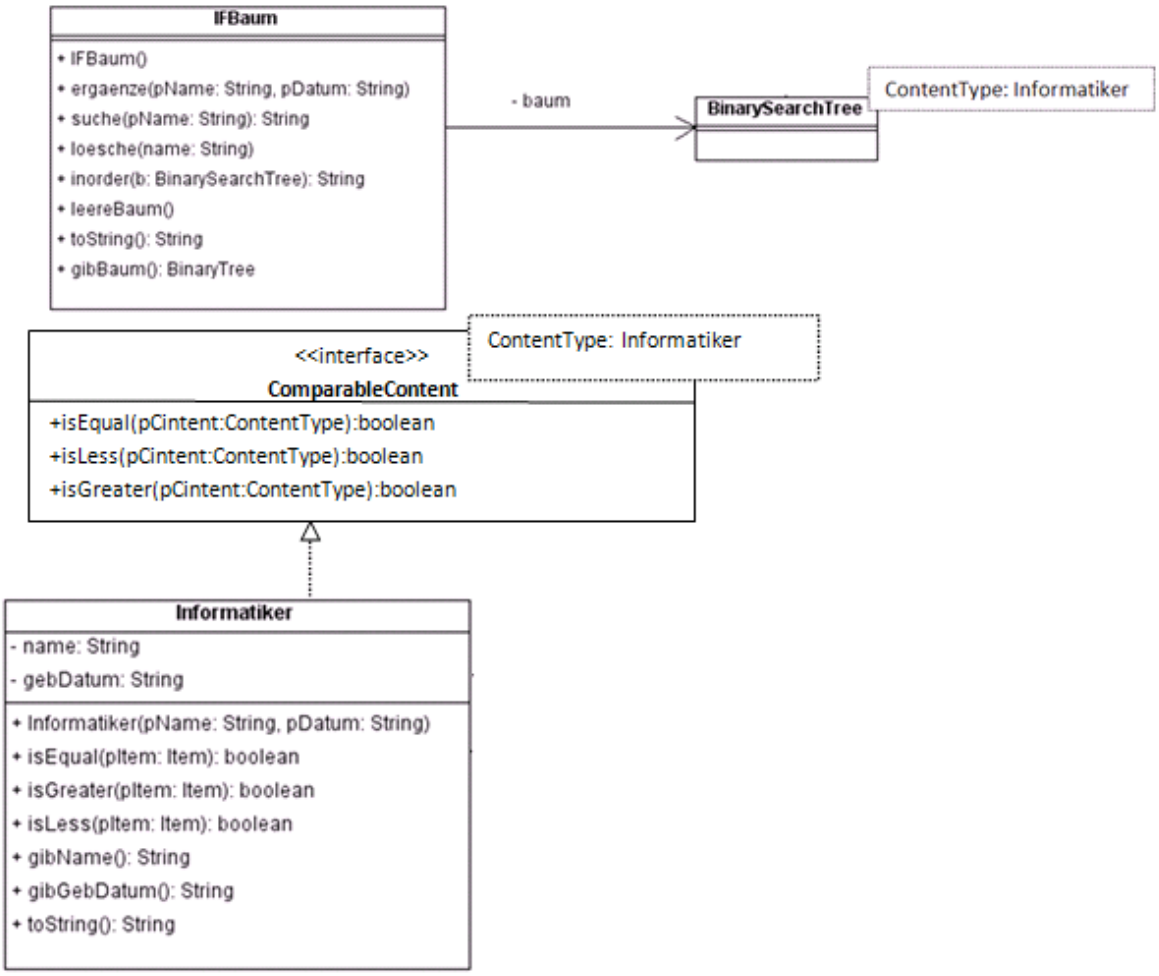
Der Informatiker-Baum als Suchbaum

Der Problemkontext Informatiker-Baum wird wieder aufgegriffen.
In einem Binärbaum sollen die Namen und die Geburtsdaten von Informatikern abgespeichert werden. Die Knoten im Baum sind nach den Namen lexikographisch geordnet.
Hier wird nun die generische Klasse **`BinarySearchTree`** zur Verwaltung der Informatikerdaten benutzt.
Es soll nach Namen gesucht werden und es wird der Name und das Geburtsdatum zurückgeliefert werden. Außerdem sollen Daten ergänzt werden. Die Daten im Baum sollen nach Namen alphabetisch sortiert ausgegeben werden.
Exemplarisch sollen folgende Funktionalitäten erfüllt sein:

- Einfügen der Informatiker-Daten in den Baum.
- Suchen nach einem Informatiker über den Schlüssel Name.
- Löschen von Informatiker-Daten über den Schlüssel Name.
- Ausgabe des kompletten Datenbestands in nach Namen sortierter Reihenfolge.



Modifiziert aus den Materialien der Federführungsgruppe Informatik NRW September 2013

Realisierung des Informatiker-Baums mithilfe der Klasse B**inarySearchTree<ContentType>**

Mithilfe der generischen Klasse **`BinarySearchTree`** können beliebig viele Objekte des Typs **`ContentType`** in einem Binärbaum (binärer Suchbaum) entsprechend einer Ordnungsrelation verwaltet werden.

Ein Objekt der Klasse **`BinarySearchTree`** stellt entweder einen leeren Baum dar oder verwaltet ein Inhaltsobjekt vom Typ **`ContentType`** sowie einen linken und einen rechten Teilbaum, die ebenfalls Objekte der Klasse **`BinarySearchTree`** sind.
Die Klasse der Objekte, die in dem Suchbaum verwaltet werden sollen, muss das generische Interface **`ComparableContent`** implementieren. Dabei muss durch Überschreiben der drei Vergleichsmethoden **`isLess, isEqual, isGreater`** (s. Dokumentation des Interfaces) eine eindeutige Ordnungsrelation festgelegt sein.

Alle Objekte im linken Teilbaum sind kleiner als das Inhaltsobjekt des Binärbaumes. Alle Objekte im rechten Teilbaum sind größer als das Inhaltsobjekt des Binärbaumes.
Diese Bedingung gilt auch in beiden Teilbäumen.

Die Klasse **`BinarySearchTree`** ist keine Unterklasse der Klasse **`BinaryTree`**, so dass deren Methoden nicht zur Verfügung stehen.

```
6  public class Informatiker implements ComparableContent<Informatiker> {
7    private String name;
8    private String gebDatum;
9
10   public Informatiker(String pName, String pDatum) {
11     super();
12     name = pName;
13     gebDatum = pDatum;
14   }
15
16   public String gibName() {
17     return name;
18   }
19
20   public String gibGebDatum() {
21     return gebDatum;
22   }
23
24   public boolean isLess(Informatiker pContent) {
25     return this.gibName().compareTo(pContent.gibName())<0;
26   }
27
28   public boolean isEqual(Informatiker pContent) {
29     return this.gibName().compareTo(pContent.gibName())==0;
30   }
31
32   public boolean isGreater(Informatiker pContent) {
33     return this.gibName().compareTo(pContent.gibName())>0;
34   }
35
36   public String toString() {
37     return name + " *" + gebDatum;
38   }
39
40 }
```


Modifiziert aus den Materialien der Federführungsgruppe Informatik NRW September 2013

Die Klasse **`Informatiker`** implementiert das Interface **`Comparable`**.

Suchen im Informatiker-Baum

```
7
8   public String suche(String pName) {
9     Informatiker inf = new Informatiker(pName,"");
10    if (!baum.isEmpty()) {
11      Object ergebnis = baum.search(inf);
12      if (ergebnis != null)
13        return ergebnis.toString() + "\n";
14      else
15        return "--\n";
16    } else {
17      return "Baum leer \n";
18    }
19  }
```

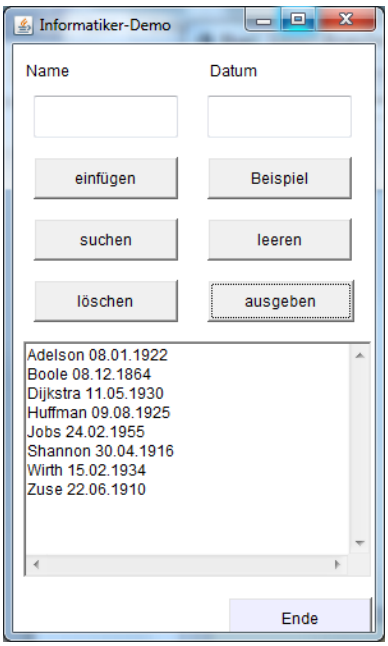
Einfügen und Löschen

```
24  public void ergaenze(String pName, String pDatum) {
25    baum.insert(new Informatiker(pName,pDatum));
26  }
27
28  public void loesche(String name) {
29    Informatiker inf = new Informatiker(name,"");
30    if (!baum.isEmpty())
31      baum.remove(inf);
32  }
```

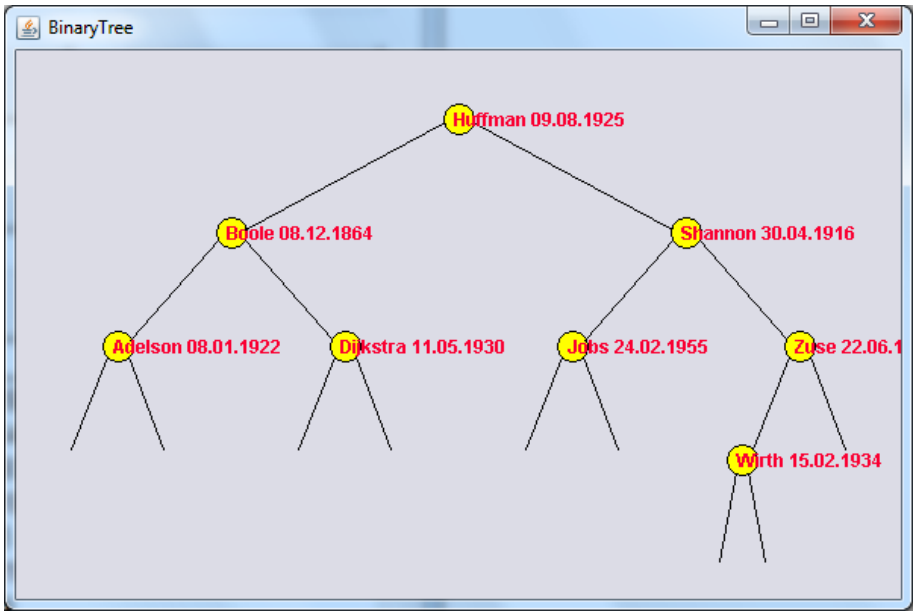
Sortierte Ausgabe

```
40  public String toString() {
41    return inorder(baum);
42  }
43
44  public String inorder(BinarySearchTree<Informatiker> b) {
45    String links = "";
46    String mitte = "";
47    String rechts = "";
48    if (!b.isEmpty()) {
49      links = inorder(b.getLeftTree());
50      mitte = b.getContent().toString()+"\n";
51      rechts = inorder(b.getRightTree());
52    }
53    return links + mitte + rechts;
54  }
```

Benutzungsoberfläche (mit zusätzlichen, schon implementierten Möglichkeiten, die das Testen erleichtern)



Zusatz: Darstellung des Binärbaums



Modifiziert aus den Materialien der Federführungsgruppe Informatik NRW September 2013