Erläuterungen zur Schnittstelle `BinarySearchTree`

Die Schnittstelle des Zentralabiturs besteht aus zwei Klassen:

- `BinarySearchTree` ist der eigentliche binäre Suchbaum.
- `ComparableContent`: In einen `BinarySearchTree` können nur Objekte eingefügt werden, die das **Interface** `ComparableContent` implementieren. `ComparableContent` erzwingt die Implementierung der Methoden `isEqual`, `isGreater` und `isLess`. Dadurch wird festgelegt, wie Objekte in den Suchbaum einsortiert werden.

Beispiel: Verwendung von `BinarySearchTree` und `ComparableContent`

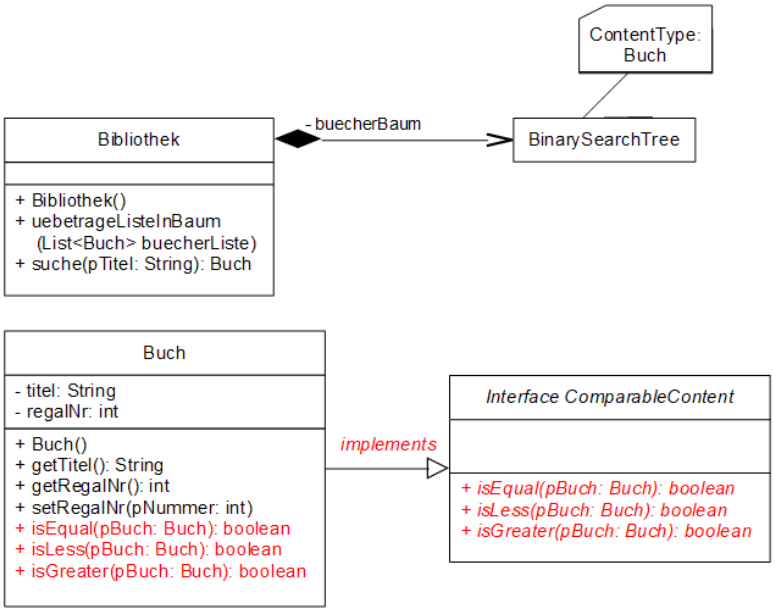
In einer Bibliothek soll der Buchbestand in einem Binären Suchbaum gespeichert werden, damit man schnell nach einem bestimmten Buchtitel suchen kann. Das heißt, das Ordnungskriterium für den Suchbaum ist die alphabetische Ordnung nach dem Titel der Bücher.

Im Detail sind für die Bibliothek folgende Methoden vorgesehen:

- **suchen:** Man übergibt einen Buchtitel und erhält das Buch
- **buecherListeEinfuegen:** Man übergibt eine Liste von Büchern, die - gemäß dem Ordnungskriterium Titel - richtig in den binären Suchbaum eingefügt werden.

Für die einzelnen Bücher werden lediglich Titel und Regalnummer gespeichert; die Regalnummer soll man nachträglich ändern können, den Titel nicht. Objekte der Klasse `Buch` sollen in einen binären Suchbaum eingefügt bzw. nach bestimmten Titeln gesucht werden. Dabei soll das Ordnungskriterium die alphabetische Ordnung nach dem Titel sein.

Implementationdiagramm



Erläuterungen:

* Bibliothek hat (=besitzt) einen `BinarySearchTree`, in dem nur Objekte vom Typ `Buch` gespeichert werden können.

* `Buch` muss das Interface `ComparableContent` implementieren, damit es überhaupt im `BinarySearchTree` `buecherBaum` gespeichert werden kann.

* Das hat als Konsequenz: `Buch` muss die Methoden `isEqual`, `isGreater` und `isLess` so überschreiben, dass die Bücher nach dem Titel verglichen werden.

Implementierung

```
public class Buch implements ComparableContent<Buch>{
  private String titel;
  private int regalNr;

  // *** Konstruktor und get-Methoden ***

  public boolean isEqual(Buch pContent) {
     return (titel.equals(pBuch.getTitel());
  }

  public boolean isLess(Buch pContent) {
     return(titel.compareTo(pBuch.getTitel())<0);
  }

  public boolean isGreater(Buch pContent) {
     return(titel.compareTo(pBuch.getTitel())>0);
  }
}
```

Jetzt können z.B. in einer Klasse `Bibliothek` Objekte der Klasse `Buch` in einen `BinarySearchTree` eingefügt werden:

```
public class Bibliothek{

   // hier werden die Buecher gespeichert
   BinarySearchTree<Buch> buecherBaum;

   public Bibliothek(){
     buecherBaum = new BinarySearchTree<Buch>();
   }

   public void uebertrageListeInBaum(List<Buch> buecherListe){
     for(buecherListe.toFirst(); buecherListe.hasAccess(); buecherListe.next()){
        Buch aktuellesBuch = buecherListe.getContent();
        buecherBaum.insert(aktuellesBuch);
     }
   }

   public Buch suche(String pTitel){
     // man muss erst ein Dummy-Buch erzeugen
     // nach dem kann man dann suchen
     Buch dummyBuch = new Buch(pTitel, -1);
     Buch ergebnis = buecherBaum.search(dummyBuch);
     return ergebnis;
   }
 }
```

Erläuterungen zur Methode `public Buch suche(String pTitel)`:

- In Objekten vom Typ `BinarySearchTree<Buch>` kann man nur nach Objekten vom Typ `Buch` suchen.
- D.h. man muss erst ein `dummyBuch` erstellen, das den gewünschten Titel trägt.
- Da `Buch` das Interface `ComparableContent` implementiert, kann man mithilfe von `dummyBuch` die Methode `search` aufrufen.
- `search` gibt dann das "richtige" gesuchte Buch zurück.
- **Wichtig:** `ergebnis` ist das "richtige" Buch; `dummyBuch` wurde nur dafür erstellt, damit man nach einem Titel suchen kann!