Human-Computer Interaction Group

Leibniz Universität Hannover

Vorlesung:
Priv.-Doz. Dr.-Ing. Matthias Becker
eMail: xmb@hci.uni-hannover.de

Übungsbetrieb:
Patric Plattner
eMail: patric.plattner@hci.uni-hannover.de

Aufgabenblatt vom **08.05.2019**
Abgabe bis **14.05.2019, 23:59**
Korrektur ab **16.05.2019**

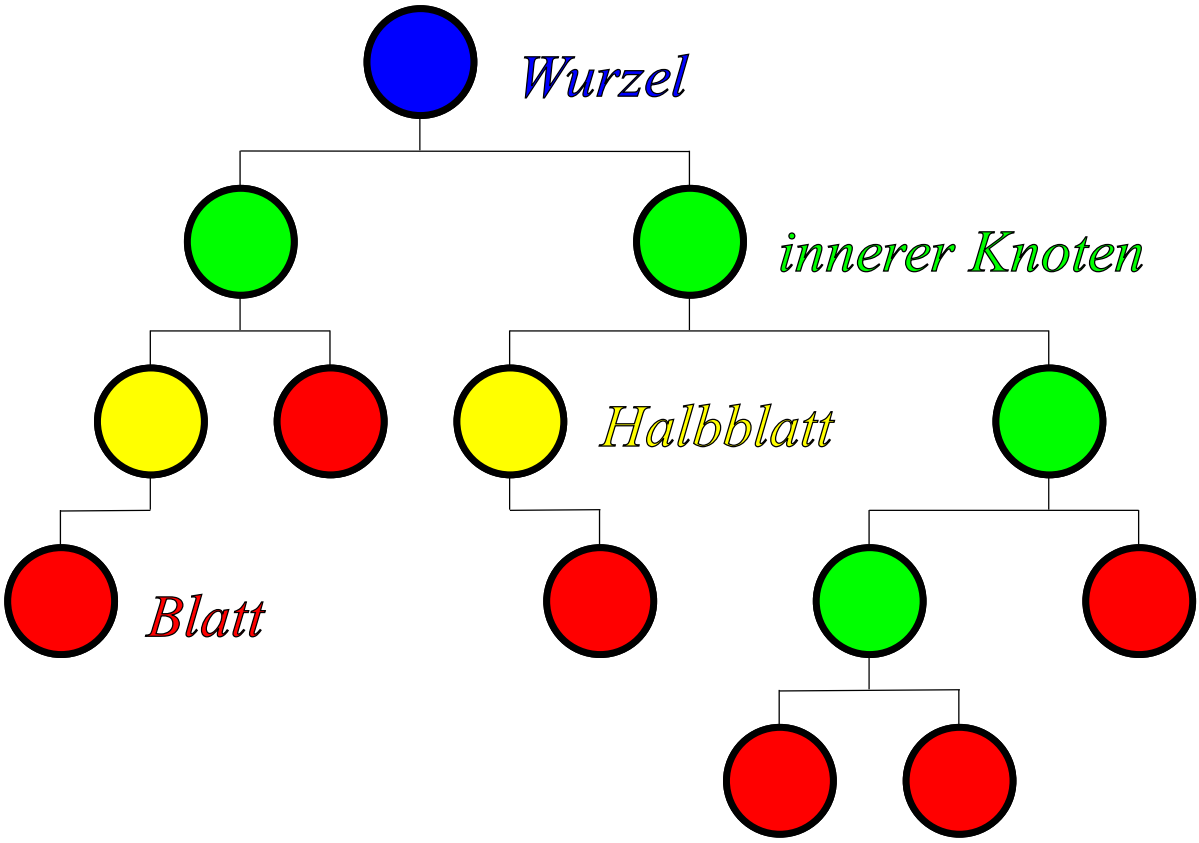
4. Hausübung

Vererbung

In diesem Aufgabenblatt lernt ihr, wie ihr konkrete Vererbung verwendet und warum das sinnvoll ist. Dazu implementieren wir einen Binären Baum fuer Integer. Dann werden wir Vererbung nutzen um einen sortierten Binären Suchbaum zu erstellen.

Binäre Bäume

Binäre Bäume sind eine recht simple Datenstruktur. Ein Knoten in einem Binären Baum hat einen Inhalt und zwei Kinder, ein linkes und ein rechtes. Den obersten Knoten nennt man Wurzel. Einen Knoten, der Kinder hat, nennt man internen Knoten. Ein Knoten, der keine Kinder hat, nennt man Blatt. Ein Binärer Baum kann wie folgt dargestellt werden:



Beispielbaum (Quelle: **https://commons.wikimedia.org/wiki/File:Binary-tree.svg** unter **Creative Commons Attribution-Share Alike 3.0 Germany** Lizenz)

In so einen Baum können wir nun Knoten einsetzen, löschen oder nach Knoten suchen. Allerdings müssen wir uns Gedanken machen, wie wir einen solchen Baum aufbauen. Man baut so einen Baum durch das Einsetzen von Werten auf. So könnte man zunächst eine 1,

Alle Abgaben sind bis zu dem in der Kopfzeile vermerkten Datum in unserem UploadTool abzugeben. Eine Anleitung zum Hochladen findet ihr hier. Prüft ob eure Abgabe ordentlich hochgeladen wurde, indem ihr sie selber nach dem Hochladen runterladet. Solltet ihr es nicht schaffen, innerhalb des vorgegebenen Zeitraumes die Lösung korrekt abzugeben, wird dies zum Nichtbestehen der Studienleistung führen. Bei technischen Problemen mit dem UploadTool bitte eine Mail an patric.plattner@hci.uni-hannover.de schicken.

Vorlesung:
Priv.-Doz. Dr.-Ing. Matthias Becker
eMail: xmb@hci.uni-hannover.de

Übungsbetrieb:
Patric Plattner
eMail: patric.plattner@hci.uni-hannover.de

Aufgabenblatt vom **08.05.2019**
Abgabe bis **14.05.2019, 23:59**
Korrektur ab **16.05.2019**

dann eine 2 und dann eine 3 einsetzen und hätte dann einen Baum, in dem diese drei Zahlen als Knoten drin sind.

Allerdings stellt sich dann die Frage, *wo* diese Knoten im Baum sind, wenn wir danach suchen wollen. Das kommt darauf an, wie die Knoten eingesetzt werden. Es gibt verschiedene Methoden dafür, aber wir schauen uns zunächst eine recht simple Methode zum einsetzen an: zufälliges Einsetzen. Diese Methode ist an sich nicht wirklich gut, aber sie eignet sich gut als eine Art Basismethode. Dazu starten wir bei der Wurzel und entscheiden zufällig, ob wir links oder rechts einsetzen wollen. Sollte in der gewählten Richtung kein Kind sein (also null), dann setzen wir dort einen neuen Knoten mit dem gegebenen Wert ein. Sollte dort bereits ein Knoten sein, so wiederholen wir dasselbe für diesen Knoten solange, bis wir ein leeres Feld zum Einsetzen finden.

Nun wissen wir, wie in unseren Baum eingefügt wird, allerdings wissen wir, da das Einfügen zufällig ist, noch nicht, wie wir gezielt nach einem Wert suchen können. Deshalb müssen wir einfach den ganzen Baum ablaufen und hoffen, dass wir den Wert irgendwann finden. Dazu eignet sich ein so genannter InOrder Durchlauf sehr gut. Ein InOrder Durchlauf funktioniert wie folgt:

```
algorithm inOrder(IntBinTree treeNode) returns String
      result += inOrder fuer treeNode.left, falls left existiert;
      result += treeNode.content
      result += inOrder fuer treeNode.right, falls right existiert;
      return result
```

Wir koennen diesen Algorithmus nun so anpassen, dass wir stattdessen nach einem Element suchen:

```
algorithm inOrderSearch(IntBinTree treeNode, int toSearch) returns boolean
      rufe inOrderSearch fuer den linken Teilbaum auf und gebe true zurueck,
wenn inOrderSearch true zurueckgibt
      falls treeNode.content == toSearch, gebe true zurueck
      rufe inOrderSearch fuer den rechten Teilbaum auf und gebe true zurueck,
wenn inOrderSearch true zurueckgibt
      gebe false zurueck.
```

Mit diesem Algorithmus koennen wir nach einem Element in einem Binärbaum suchen. Diese Art und Weise einen Baum aufzubauen und zu durchsuchen ist zwar nicht effizient, aber sie funktioniert schonmal.

Vorlesung:
Priv.-Doz. Dr.-Ing. Matthias Becker
eMail: xmb@hci.uni-hannover.de

Übungsbetrieb:
Patric Plattner
eMail: patric.plattner@hci.uni-hannover.de

Aufgabenblatt vom **08.05.2019**
Abgabe bis **14.05.2019, 23:59**
Korrektur ab **16.05.2019**

Binäre Suchbäume

Zwar funktioniert das zufällige Einsetzen in einen Baum als Methode zum Einfügen, aber es erschwert das Suchen, da wir nicht wissen, wie der Baum aufgebaut ist. Für diesen Zweck gibt es binäre Suchbäume.

Ein binärer Suchbaum hat die Eigenschaft, dass alle Werte im linken Teilbaum eines Knotens kleiner sind und alle im rechten Teilbaum grösser. Dies erfordert zwar Mehraufwand/Vergleiche beim Einfügen, hat aber den Vorteil, dass wir zielgerichtet und effizient im Baum suchen können. Das Einfügen in einen Suchbaum funkioniert wie folgt:

```
algorithm insert(IntBinTree tree, int toInsert)
	if toInsert > tree.content then
		insert(tree.right, toInsert)
	else if toInsert < tree.content then
		insert(tree.left, toInsert)
```

Somit wird das Suchen im Baum trivial:

```
algorithm search(IntBinTree tree, int search) returns boolean
	if tree.content == search then
		return true
	else if tree.content > search then
		if tree.left == null then
			return false
		else
			search(tree.left, search)
	else
		if tree.right == null then
			return false
		else
			search(tree.right, search)
```

Umsetzung in Java

An sich könnte man einfach zwei Klassen schreiben, die beide dieselben Methoden haben, nur dass sich die `insert()` und `search()` Methoden unterscheiden, allerdings gibt es in Java einen Mechanismus um solche Beziehungen sinnvoll zu modellieren. Wenn man nämlich genauer ueberlegt, ist ein binärer Suchbaum nur eine spezielle Version eines normalen binären Baumes. Selbst viele der Methoden überschneiden sich. Getter, Setter

Vorlesung:
Priv.-Doz. Dr.-Ing. Matthias Becker
eMail: xmb@hci.uni-hannover.de

Übungsbetrieb:
Patric Plattner
eMail: patric.plattner@hci.uni-hannover.de

Aufgabenblatt vom **08.05.2019**
Abgabe bis **14.05.2019, 23:59**
Korrektur ab **16.05.2019**

und die Baumdurchläufe sind sogar exakt dieselben. Für solche Fälle gibt es in Java die sogenannte *konkrete Vererbung*. Eine Klasse, die von einer sogenannten Superklasse erbt, erbt alle Methoden und Felder der Superklasse und kann zusätzlich eigene Methoden und Felder hinzufügen und somit die Superklasse erweitern, oder bereits vorhandene Methoden in der Superklasse überschreiben und somit das Verhalten ändern.

Um diese Art von Vererbung mit binärem Baum und binärem Suchbaum umzusetzen, implementieren wir zunächst die Klasse `IntBinTree`. Diese soll folgende Methoden implementieren:

- `IntBinTree getLeft()`
- `IntBinTree getRight()`
- `void setLeft(IntBinTree t)`
- `void setRight(IntBinTree t)`
- `int getContent()`
- `String inOrder()`
- `boolean search(int i) //gibt true zurueck falls i in baum`
- `void insert(int i)`

Diese sollen sich so verhalten, wie in der Erklärung zu Binärbaumen angedeutet. Hier soll das zufällige Einsetzen und die dazu passende Suche implementiert werden. Implementiere zusätzlich einen passenden Konstruktor. Zusätzlich wollen wir die `Object.toString()` Methode überschreiben, so dass sie einen String ausgibt, der wie folgt aussieht:

`(left.toString() this.getContent() right.toString())`

Sollte ein Kind null sein, so ersetze `child.toString()` mit `_`. Falls der Aktuelle Knoten ein Blatt ist, soll anstatt von `(_ x _)` einfach nur `x` ausgegeben werden.

Nun implementieren wir den binären Suchbaum. Erstelle dafür eine Klasse `IntSearchTree`, die von `IntBinTree` erbt. Überschreibe dann die Methoden `insert()` und `search()`, so dass sie wie in der Erklärung zu Suchbäumen beschrieben funktionieren. Erstelle auch hier einen passenden Konstruktor.

Aufgabe

Um zu Bestehen müsst ihr die beiden Klassen `IntBinTree` und `IntSearchTree` so wie beschrieben implementieren. Die geforderten Methoden sind:

- `IntBinTree getLeft()`
- `IntBinTree getRight()`

Vorlesung:
Priv.-Doz. Dr.-Ing. Matthias Becker
eMail: xmb@hci.uni-hannover.de

Übungsbetrieb:
Patric Plattner
eMail: patric.plattner@hci.uni-hannover.de

Aufgabenblatt vom **08.05.2019**
Abgabe bis **14.05.2019, 23:59**
Korrektur ab **16.05.2019**

- `void setLeft(IntBinTree t)`
- `void setRight(IntBinTree t)`
- `int getContent()`
- `String inOrder()`
- `String toString()`
- `boolean search(int i)`
- `void insert(int i)`

Arbeite dabei mit Packages und halte dich bei der Package Benennung an die Oracle Konvention, also soll das Package so aussehen: `de.uni_hannover.hci.name`. Ergänze deinen Code mit sinnvollen Kommentaren und füge für Methoden und Klassen Javadoc Kommentare hinzu, wo es sinnvoll ist.

Erstellt anschliessend eine Main Klasse mit Main Methode, in der ihr zwei Variablen vom Typ `IntBinTree` erstellt. Dann erstellt ihr dort jeweils einen `new IntBinTree(10, null, null)` und einen `new IntSearchTree(10, null, null)`. Setzt dann nacheinander in beide Bäume die Werte 3, 6, 16, 32, 7, 2, 54 und 15 ein. Gebt danach beide Bäume auf der Konsole aus und gebt für beide den inOrder Durchlauf aus.