**LK-Q1-V**

**3. Die Datenstruktur Binärbaum als Spezialfall eines Graphen im Anwendungskontext unter Nutzung der Klasse BinaryTree<ContentType>**

1. Definition eines Binärbaumes und grundlegende Begriffe
2. Erarbeitung der Klasse BinaryTree und beispielhafte Anwendung der Operationen

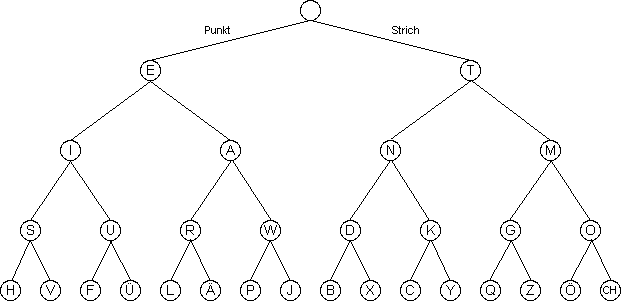
**Aufgabe**

*Modellieren und implementieren Sie ein Java-Programm, das in der Lage ist, eine Zeichenfolge aus Großbuchstaben in das Morsealphabet zu codieren und eine Folge von Morsezeichen, die durch „/“ getrennt sind, zu dekodieren.*

**Das Morsealphabet**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| A ·- | F ··-· | K -·- | P ·--· | U ··- | Z --·- |
| B -··· | G --· | L ·-·· | Q --·· | V ···- | Ä ·-·- |
| C -·-· | H ···· | M -- | R ·-· | W ·-- | Ö ---· |
| D -·· | I ·· | N -· | S ... | X -··- | Ü ··-- |
| E · | J ·--- | O --- | T - | Y -·-- | CH ---- |

Übersichtlich lässt sich der Morsecode in folgender Struktur darstellen.



Bei dieser Struktur handelt es sich um einen Baum (zusammenhängender Graph ohne Zyklen), der einen Anfangsknoten hat und von dessen Knoten eine, zwei oder keine Kante ausgehen. Diese Datenstruktur heißt **Binärbaum**.

**Definition 1**

Ein Binärbaum ist ein gerichteter Baum mit einem ausgezeichneten Knoten, Wurzel genannt, in dem von jedem Knoten höchsten zwei Kanten ausgehen. Die Nachfolgeknoten jedes Knotens werden Kinder genannt. Es wird zwischen rechtem und linkem Kind unterschieden.

Die Knoten, die keine Kinder haben, heißen **Blätter**.

Beispiel: In dem Morse-Binärbaum hat die Wurzel keinen Inhalt. Der Knoten S ist das linke und der Knoten U das rechte Kind des Knotens I. Blätter sind die Knoten H, V, F, ...

Mit den Klassen Graph, Edge und Vertex ließen sich mit Erweiterungen durch Unterklassen auch Binärbäume verwalten.

Die Datenstruktur Binärbaum lässt sich aber auch unabhängig vom Graphen wie folgt rekursiv definieren.

**Definition 2**

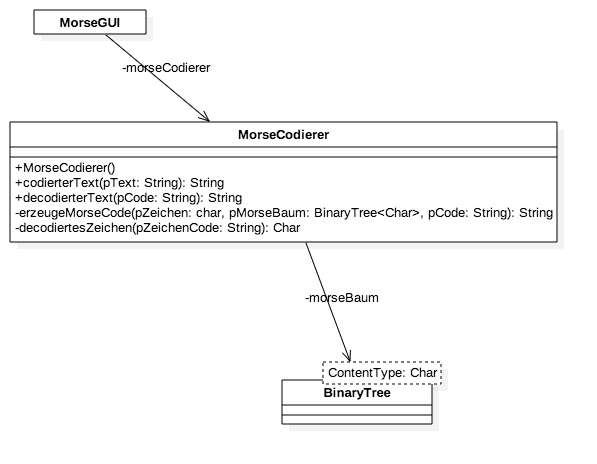
Ein **Binärbaum** ist entweder leer oder er besteht aus einer Wurzel und einem linken und rechten Binärbaum, Teilbaum genannt.

Danach sind Blätter also Binärbäume, die zwei leere Teilbäume haben.

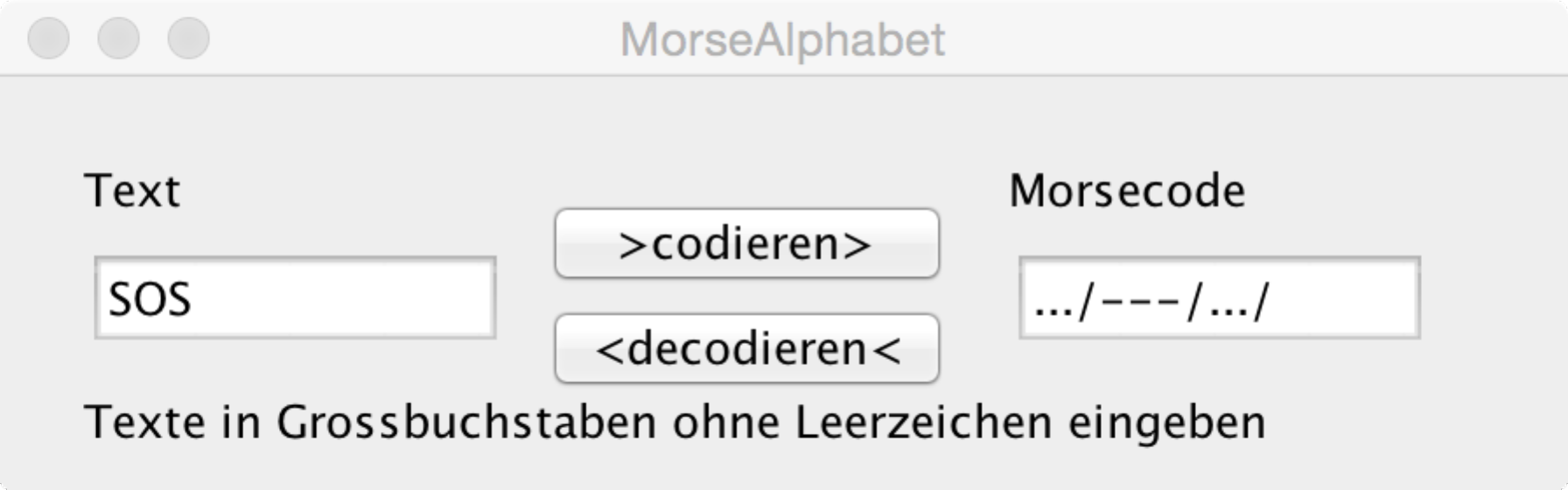
Zur Erzeugung und Verwaltung von Binärbaum-Objekten steht die generische Klasse BinarySearchTree<ContentType> zur Verfügung, die sich an der rekursiven Definition orientiert.

### Dokumentation Klasse BinaryTree<ContentType> s. Anhang

**Implementationsdiagramm für den MorseCodiererDecodierer**



**Benutzungsschnittstelle**



Der Konstruktur MorseCodierer()der Klasse MorseCodierer erzeugt den Morsebaum. Um den ASCII-Code verwenden zu können, wurde auf die Umlaute und das Zeichen „CH“ verzichtet. Man beachte die Verwendung verschiedener Konstruktoren der Klasse BinaryTree.

public MorseCodierer() {

BinaryTree<Character> lBaum4Links = new BinaryTree<Character>(new Character('H'));

BinaryTree<Character> lBaum4Rechts = new BinaryTree<Character>(new Character('V'));

BinaryTree<Character> lBaum3Links = new BinaryTree<Character>(new Character('S'),

lBaum4Links, lBaum4Rechts);

lBaum4Links = new BinaryTree<Character>(new Character('F'));

lBaum4Rechts = new BinaryTree<Character>();

BinaryTree<Character> lBaum3Rechts = new BinaryTree<Character>(new Character('U'),

lBaum4Links, lBaum4Rechts);

BinaryTree<Character> lBaum2Links = new BinaryTree<Character>(new Character('I'),

lBaum3Links, lBaum3Rechts);

lBaum4Links = new BinaryTree<Character>(new Character('L'));

lBaum4Rechts = new BinaryTree<Character>();

lBaum3Links = new BinaryTree<Character>(new Character('R'), lBaum4Links,

lBaum4Rechts);

lBaum4Links = new BinaryTree<Character>(new Character('P'));

lBaum4Rechts = new BinaryTree<Character>(new Character('J'));

lBaum3Rechts = new BinaryTree<Character>(new Character('W'), lBaum4Links,

lBaum4Rechts);

BinaryTree<Character> lBaum2Rechts = new BinaryTree<Character>(new Character('A'),

lBaum3Links, lBaum3Rechts);

BinaryTree<Character> lBaum1Links = new BinaryTree<Character>(new Character('E'),

lBaum2Links, lBaum2Rechts);

…

lBaum4Links = new BinaryTree<Character>(new Character('G'));

lBaum4Rechts = new BinaryTree<Character>();

lBaum3Rechts = new BinaryTree<Character>(new Character('O'), lBaum4Links,

lBaum4Rechts);

lBaum2Rechts = new BinaryTree<Character>(new Character('M'),

lBaum3Links, lBaum3Rechts);

BinaryTree<Character> lBaum1Rechts = new BinaryTree<Character>(new Character('T'),

lBaum2Links, lBaum2Rechts);

morseBaum = new BinaryTree<Character>(new Character(' '), lBaum1Links,

lBaum1Rechts);

}

Die Methode codierterText liefert den in den Morsecode ü*b*ersetzten Text pText. Die von ihr aufgerufene private Methode erzeugeMorsecodesucht den Morsecode eines einzelnen Zeichens im Morsebaum.

public String codierterText (String pText) {

int lZaehler = 0;

String lCode = "";

while ((lZaehler < pText.length()) && (pText.charAt (lZaehler) >= 'A') &&

(pText.charAt (lZaehler) <= 'Z')) {

lCode = lCode + erzeugeMorsecode (pText.charAt(lZaehler), morseBaum, "") + "/";

lZaehler++;

}

return lCode;

}

private String erzeugeMorsecode (char pZeichen, BinaryTree<Character> pMorseBaum,

String pCode) {

if (!pMorseBaum.isEmpty()){

if (pZeichen == pMorseBaum.getContent()){

return pCode;

}

else {

String lCodeLinkerTeilbaum = erzeugeMorsecode(pZeichen,

pMorseBaum.getLeftTree(), pCode+".");

if (lCodeLinkerTeilbaum.equals(""))

return erzeugeMorsecode(pZeichen, pMorseBaum.getRightTree(), pCode+"-");

else

return lCodeLinkerTeilbaum;

}

}

else

return "";

}

Die Methode decodierterText liefert die Decodierung des Morsecodes pCode. Die von ihr aufgerufene private Methode decodiertesZeichendecodiert ein einzelnes Zeichen mit Hilfe des Morsebaums.

public String decodierterText (String pCode) {

String lText = "";

String lMorsezeichen;

if(pCode.charAt(pCode.length()-1) != '/');

pCode = pCode + '/';

do {

lMorsezeichen=pCode.substring(0, pCode.indexOf('/'));

pCode = pCode.substring(pCode.indexOf('/')+1,pCode.length());

if (lMorsezeichen != "") {

lText=lText+decodiertesZeichen(lMorsezeichen);

}

} while (!pCode.equals(""));

return lText;

}

public Character decodiertesZeichen( String pZeichenCode){

int lZaehler = 0;

BinaryTree<Character> lBaum = morseBaum;

while (lZaehler < pZeichenCode.length()) {

if (pZeichenCode.charAt(lZaehler) == '.')

lBaum = lBaum.getLeftTree();

else

lBaum = lBaum.getRightTree();

lZaehler++;

}

return lBaum.getContent();

}

**Anhang**

**Dokumentation der Klasse BinaryTree<ContentType>**

Mithilfe der generischen Klasse **BinaryTree** können beliebig viele Objekte vom Typ **ContentType** in einem Binärbaum verwaltet werden. Ein Objekt der Klasse stellt entweder einen leeren Baum dar oder verwaltet ein Inhaltsobjekt sowie einen linken und einen rechten Teilbaum, die ebenfalls Objekte der generischen Klasse **BinaryTree** sind.

|  |  |
| --- | --- |
| **Konstruktor** | **BinaryTree<ContentType>()** |
|  | Nach dem Aufruf des Konstruktors existiert ein leerer Binärbaum. |
|  |  |
| **Konstruktor** | **BinaryTree<ContentType>(ContentType pContent)** |
|  | Wenn der Parameter pContent ungleich null ist, existiert nach dem Aufruf des Konstruktors der Binärbaum und hat pContent als Inhaltsobjekt und zwei leere Teilbäume. Falls der Parameter null ist, wird ein leerer Binärbaum erzeugt. |
|  |  |
| **Konstruktor** | **BinaryTree<ContentType>(ContentType pContent,**  **BinaryTree<ContentType> pLeftTree,**  **BinaryTree<ContentType> pRightTree)** |
|  | Wenn der Parameter pContent ungleich null ist, wird ein Binärbaum mit pContent als Inhaltsobjekt und den beiden Teilbäume pLeftTree und pRightTree erzeugt. Sind pLeftTree oder pRightTree gleich null, wird der entsprechende Teilbaum als leerer Binärbaum eingefügt. Wenn der Parameter pContent gleich null ist, wird ein leerer Binärbaum erzeugt. |
|  |  |
| **Anfrage** | **boolean isEmpty()** |
|  | Diese Anfrage liefert den Wahrheitswert true, wenn der Binärbaum leer ist, sonst liefert sie den Wert false. |
|  |  |
| **Auftrag** | **void setContent(ContentType pContent)** |
|  | Wenn der Binärbaum leer ist, wird der Parameter pContent als Inhaltsobjekt sowie ein leerer linker und rechter Teilbaum eingefügt. Ist der Binärbaum nicht leer, wird das Inhaltsobjekt durch pContent ersetzt. Die Teilbäume werden nicht geändert. Wenn pContent null ist, bleibt der Binärbaum unverändert. |
|  |  |
| **Anfrage** | **ContentType getContent()** |
|  | Diese Anfrage liefert das Inhaltsobjekt des Binärbaums. Wenn der Binärbaum leer ist, wird null zurückgegeben. |
|  |  |

|  |  |
| --- | --- |
| **Auftrag** | **void setLeftTree(BinaryTree<ContentType> pTree)** |
|  | Wenn der Binärbaum leer ist, wird pTree nicht angehängt. Andernfalls erhält der Binärbaum den übergebenen Baum als linken Teilbaum. Falls der Parameter null ist, ändert sich nichts. |
|  |  |
| **Auftrag** | **void setRightTree(BinaryTree<ContentType> pTree)** |
|  | Wenn der Binärbaum leer ist, wird pTree nicht angehängt. Andernfalls erhält der Binärbaum den übergebenen Baum als rechten Teilbaum. Falls der Parameter null ist, ändert sich nichts. |
|  |  |
| **Anfrage** | **BinaryTree<ContentType> getLeftTree()** |
|  | Diese Anfrage liefert den linken Teilbaum des Binärbaumes. Der Binärbaum ändert sich nicht. Wenn der Binärbaum leer ist, wird null zurückgegeben. |
|  |  |
| **Anfrage** | **BinaryTree<ContentType> getRightTree()** |
|  | Diese Anfrage liefert den rechten Teilbaum des Binärbaumes. Der Binärbaum ändert sich nicht. Wenn der Binärbaum leer ist, wird null zurückgegeben. |