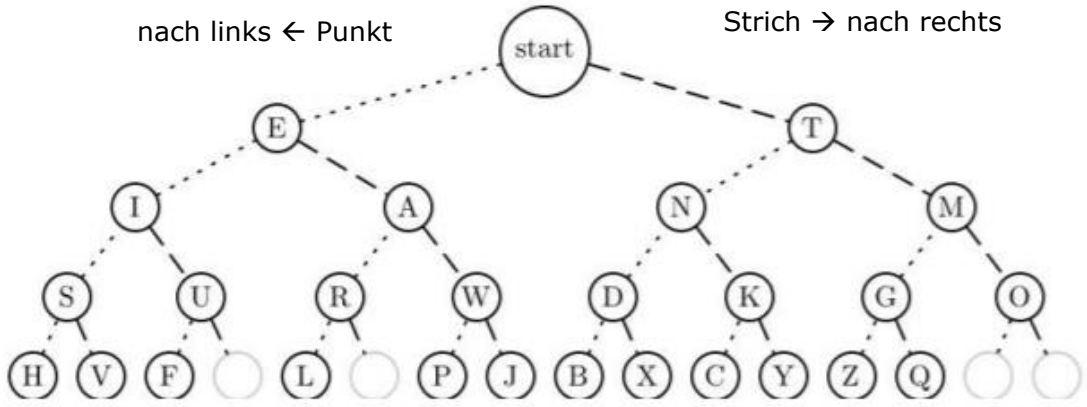
Morse Code als Baum

Einleitung:

Samuel Morse entwickelte ab 1837 den ersten elektrischen Telegrafen.
Die Zeichen wurden mit dem Morse-Code, lange und kurze Töne, codiert.

A·−	B−···	C−·−·
D−··	E·	F··−·
G−−·	H····	i··
J·−−−	K−·−	L·−··
M−−	N−·	O−−−
P·−−·	Q−−·−	R·−·
S···	T−	U··−
V···−	W·−−	X−··−
Y−·−−	Z−−··	
0−−−−−	1·−−−−	2··−−−
3···−−	4····−	5·····
6−····	7−−···	8−−−··
9−−−−·		

Da der Code aus einer Folge von Strichen und Punkten besteht, lässt er sich als Baum darstellen.



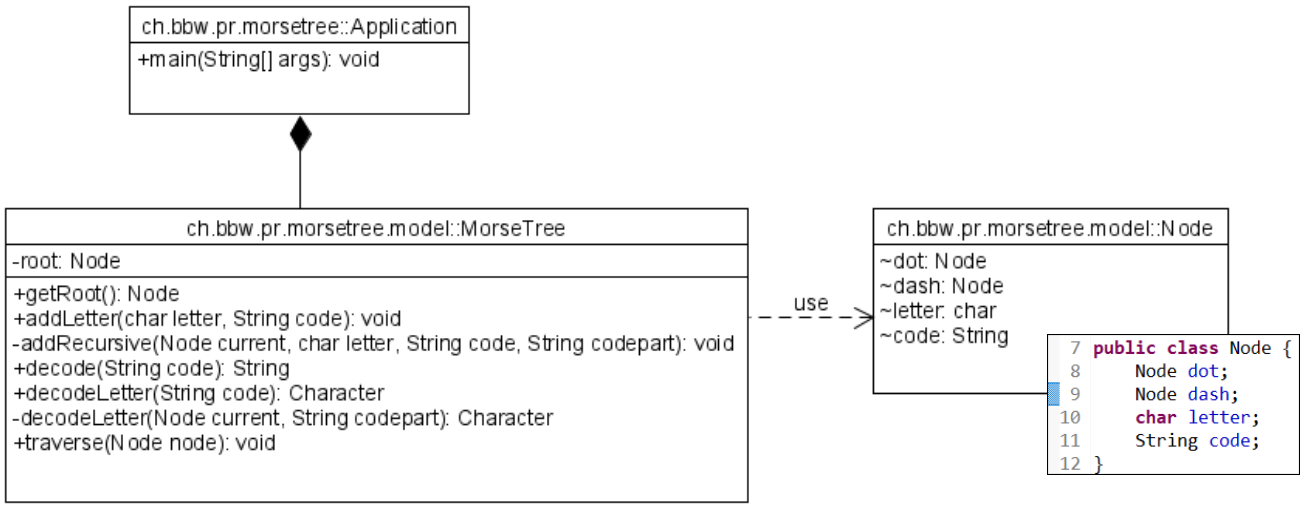
Dieser Baum wird vor allem *bei der Entschlüsselung* verwendet.

Beginnend bei der Wurzel folgt man den Punkten und Strichen ... / --- / ... und entschlüsselt so den Code.
Der Baum ist dabei für das Entschlüsseln intuitiver als eine Tabelle.

Ziel

- ⇨ Sie setzen einen Tree als Morsebaum gemäss dem Bild um.
  Und entschlüsseln mit dem Baum Morse-Nachrichten.
- ⇨ Sie werden dabei verschiedene **rekursive Methoden** anwenden:
  - *addRecursive*
  - *decodeLetter*
  - *traverse*

Idee für die Klassen



- **Application**:
  Es ist eine einfache Console Application.
  In ***main()*** werden Sie die Methoden von MorseTree aufrufen
  - den Baum mit Buchstaben befüllen
  - den Baum traversieren
  - Buchstaben entschlüsseln

- **MorseTree**
  Quasi die Business-Klasse mit den Methoden, um den Baum zu verwalten.

- **Node**
  Ein Konten im Baum, mit einem Attribut für die beiden Folge-Konten im Baum.
  (Diese Attribute können auch null sein, wenn es keine weiteren Knoten hat.)

Schritt 1: Den Baum mit den Buchstaben aufbauen

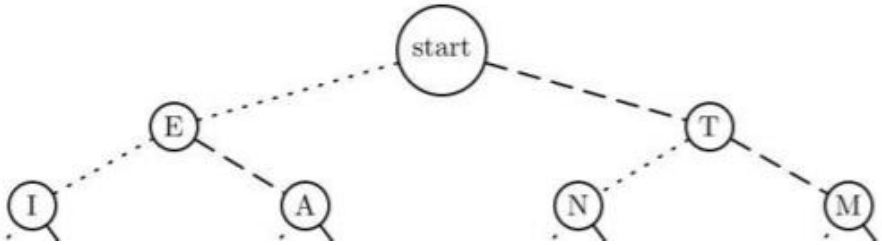
Das wollen wir erreichen:

Mit Hilfe von ***addLetter(..)*** werden Buchstaben in den Baum eingefügt.

Im ***main()*** sieht dann das so aus.:

```
16          System.out.println("Add some letters");
17          myTree.addLetter('e', ".");
18          myTree.addLetter('a', ".-");
19          myTree.addLetter('i', "..");
20          myTree.addLetter('t', "-");
21          myTree.addLetter('n', "-.");
22          myTree.addLetter('m', "--");
```

Daraus entsteht dieser Baum.



Und nun die Detail Erklärung wie das geht.

a) Zunächst das Objekt der Business-Klasse instanziieren.

```
public class Application {

 public static void main(String[] args) {
       System.out.println("Morse-Tree");
       MorseTree myTree = new MorseTree();
```

1. Der Root-Node wird im Constructor erstellt.

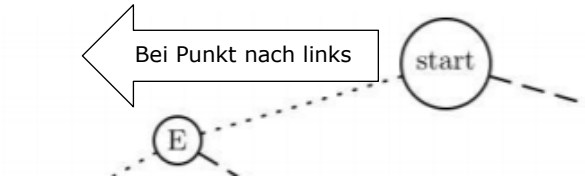
```
8   public class MorseTree {
9      private Node root;
10
11     public void MorseTree() {
12        root = new Node();
13        root.code="root";;
14     }
```

b) Nun wird in ***main()*** der erste Buchstabe eingefügt.

```
myTree.addLetter('e', ".");
```

```
20    public void addLetter(char letter, String code) {
21       if(root==null) {
22          System.out.println("MorseTree.addLetter, failed, no root Node");
23          return;
24       }
25       addRecursive(root, letter, code, code);
26    }
```

addRecursive fügt als rekursive Methode ausgehend vom Root-Node und jeweils dem Morse-Code folgend *… Punkt nach links … Strich nach rechts …* durch die Kanten des bestehenden Baumes. Bis der Knoten für den Buchstaben gefunden ist.
Auf dem Weg *fehlende* Knoten werden neu erstellt.



Man bewegt sich von Konten zu Konten. Um jedes Mal wird die rekursive Methode erneut verarbeitet.

```
private void addRecursive(Node current, char letter, String code
   , String codepart) {

   if(codepart.length()==0) {
      //codepart ist fertig ausgelesen, ich bin der Knoten
      // todo:
      // → letter im node eintragen
      // → code im node eintragen
   }
   else if(codepart.charAt(0) == '.') {
      //es ist ein Punkt.. weiter mit dem dot-Knoten
      if(current.dot == null) current.dot = new Node();
         addRecursive(current.dot, letter, code, codepart.substring(1));
      }
      else if(codepart.charAt(0) == '-') {
         //es ist ein Strich ... weiter mit dem dash-Knoten
         if(current.dash == null) current.dash = new Node();
            //todo, rekursiver Aufruf
            //addRecursive( ...
   }else {
      //unbekanntes oder Trennzeichen
      //todo, rekursiver Aufruf
      //addRecursive( ...
   }
}
```

Haben Sie den **Trick mit codepart.substring(1)** verstanden?

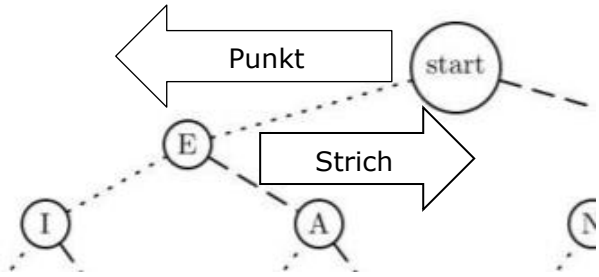
⇨ Bei jedem rekursiven Aufruf, wird das vorderste Zeichen des *codes* weggeschnitten. Der nächste Aufruf steht also nächste Zeichen zuvorderst im Code, dass dann im rekursiven Aufruf verarbeitet wird.

Hier die ganze Methode ohne die todo:

```
28  @   private void addRecursive(Node current, char letter, String code, String codepart) {
29          if(codepart.length()==0) {
30              //codepart ist fertig ausgelesen, ich bin der Knoten
31              current.letter = letter;
32              current.code = code;
33          }
34          else if(codepart.charAt(0) == '.') {
35              //weiter mit dem dot-Knoten
36              if(current.dot == null) current.dot = new Node();
37  ↺           addRecursive(current.dot, letter, code, codepart.substring( beginIndex: 1));
38          }else if(codepart.charAt(0) == '-') {
39              //weiter mit dem dash-Knoten
40              if(current.dash == null) current.dash = new Node();
41  ↺           addRecursive(current.dash, letter, code, codepart.substring( beginIndex: 1));
42          }else {
43              //unbekanntes oder Trennzeichen
44  ↺           addRecursive(current, letter, code, codepart.substring( beginIndex: 1));
45          }
46      }
```

Und weiter geht es… den nächsten Buchstaben einfügen: **a**

```
myTree.addLetter('a', ".-");
```



Den Baum ausgeben

Hat man einige Buchstaben eingetragen, dann kann man die Buchstaben und Codes der Konten ausgeben.

```
Print the tree:
  root
e .
i ..
a .-
t -
n -.
m --
```

Dabei **traversiert** man den Baum von Konten zu Konten.

Der Aufruf im main() ausgehend vom Root-Konten alle Konten ausgeben.

```
myTree.traverse(myTree.getRoot());
```

Und hier der Ansatz für die Methode *traverse:*

- Rufe vom aktuellen Knoten *traverse* für den *node.dot* und den *node.dash* auf.

```
public void traverse(Node node) {
   if (node != null) {
      System.out.println(node.letter + " " + node.code);
      //Todo
      //traverse( ...
      //traverse( ...
   }
}
```

Einen Morse-Code decodieren (umwandlen) zu einem Buchstaben

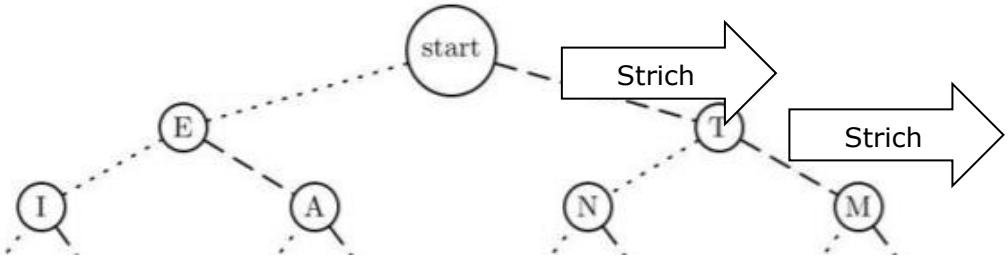
Zunächst der Aufruf im ***main()***.

```
System.out.println("decode -- " + myTree.decodeLetter("--"));
```

Die Methode decodeLetter folgt rekursiv dem Baum. Wiederum steuert der *code* ("--"), welches der nächste Knoten ist.

Die Methode ähnelt der Methode für das rekursive Einfügen.
Allerdings erstellt die Methode *decodeLetter* keine neuen Nodes und fügt nichts ein!
Dafür gibt sie, wenn der Knoten gefunden wurde (der code im rekursiven Aufruf leer ist) den ***Buchstaben*** des Node mit ***return*** zurück.

```
... myTree.decodeLetter("--") ...
```




```
//Vorbereitende Methode
public Character decodeLetter(String code) {
   if(root==null) {
      return ' ';
   }
   return decodeLetter(root, code);
}

//rekursive Methode
private Character decodeLetter(Node current, String codepart) {
   //todo: hier implementieren.
```

Einen Morse-Code aus mehreren Buchstaben in eine Text umwandeln

Dazu muss man den Morse Code zuerst in Buchstaben-Codes aufsplitten und dann die diese Codes im Baum suchen

Zunächst der Aufruf im *main()*

```
String tmp = "--/../-/-/.";
System.out.println("decode " + tmp + " ==> " + myTree.decode(tmp));
```

Und dann die Methode, die den Code aufsplitted und die bereits bestehende Methode decoceLetter aufruft.

```
public String decode(String code) {
   String retVal="";
   for(String part: code.split("/")) {
      retVal = retVal + decodeLetter(root, part) + " ";
   }
   return retVal;
}
```

```
Decode multiple characters
decode --/../-/-/. ==> m i t t e
```

Zusatz Aufgabe

Was muss man tun, damit man einen Text als Morse-Code übersetzen kann?

Implementieren Sie die Methoden und Aufrufe im main.