Dijkstra-Algorithmus

public void ***kuezesterWeg***(String ***startKnoten***, String ***zielKnoten***)

int startNummer = getKnotenNummer(startKnoten);	*(bestimme die Knotennummer des Startknotens und speichere diese unter startNummer ab)*
int zielNummer = getKnotenNummer(zielKnoten);	*(bestimme die Knotennummer des Zielknotens und speichere diese unter zielNummer ab)*
int aktuellerKnoten, neueDistanz;	*(Variablen)*
setze die Besuchtliste zurück	*(alle Indixe im Feld besucht erhalten den Wert false)*
setze das Distanz-Feld zurück	*(alle Indixe im Feld distanz erhalten den Wert **Integer.MAX_VALUE**)*
setze das kommtVon-Feld zurück	*(alle Indixe im Feld kommtVon erhalten den Wert -1)*
distanz[startNummer] = 0;	*(der Startknoten hat keine Entfernung zu sich selbst)*
kommtVon[startNummer] = startNummer;	*(am Index startNummer im Feld kommtVon wird die startNummer gespeichert; daran erkennen wir, dass es der Startknoten ist, da er keinen Vorgängerknoten besitzt)*

zähle i von 0 bis anzahlKnoten - 1

aktuellerKnoten = **minKnoten()**	*(der unbesuchte Knoten mit der minialen Distanz wird zum aktuellen Knoten)*
besucht[aktuellerKnoten] = true;	*(der aktuelle Knoten wird als besucht gesetzt)*

zähle abzweigNummer von 0 bis anzahlKnoten - 1

matrix[aktuellerKnoten][abzweignummer] > 0 *(es existiert eine Kante zwischen* aktuellerKnoten *und abzweigNummer)*
und der Knoten mit der Knotennummer abzweigNummer
wurde noch nicht besucht

wahr / **falsch**

neueDistanz = distanz[aktuellerKnoten] + matrix[aktuellerKnoten][abzweignummer];

(zu der Distanz des momentan aktuellen *Knotens wird das Kantengewicht zwischen* aktuellerKnoten *und dem Knoten* abzweignummer *addiert)*

neueDistanz < distanz[abzweigNummer]

wahr / **falsch**

distanz[abzweignummer] = neueDistanz;

(die neue Distanz ist geringer als die bisher gespeicherte und wird dadurch durch die neue Distanz ersetzt)

kommtVon[abzweigNummer] = aktuellerKnoten;

(*der Vorgängerknoten wird angepasst)*

Ausgabe der Entfernung **distanz[zielNummer]**

String weg = zielKnoten; *(in der Variablen weg wird am Ende der komplette Pfad stehen – der Pfad wird rückwärts beginnend beim Zielknoten ermittelt)*	
aktuellerKnoten = zielNummer;	
wiederhole solange aktuellerKnoten *ungleich startNummer* ist	
aktuellerKnoten = kommtVon[aktuellerKnoten];	*(der Vorgängerknoten vom momentanen aktuellen Knoten wird zum neuen aktuellen Knoten)*
weg = knotenPunkte[aktuellerKnoten].getBezeichner() + "/" + weg;	*(der Bezeichner vom neuen aktuellen Knoten wird einem / und dem bisherigen Weg vorangestellt)*
Ausgabe von den Variablen **weg**	

private int ***minKnoten***( )

int minPos = 0;	
int minWert = Integer.MAX_VALUE;	
zähle i von 0 bis anzahlKnoten - 1	
der Knoten mit der Knotennummer i wurde noch nicht besucht **und** minWert > distanz[i]	
wahr	**falsch**
minWert = distanz[i]; (*die Distanz vom Knoten mit der Knotennummer i ist kleiner als der in der Variablen minWert gespeicherte Wert)*	
minPos = i; *(der Knoten mit der Knotennummer i hat die kürzeste Entfernung zum Startknoten)*	
Rückgabe von **minPos**	