C23 – Laborübung 4

Kontext

Ein Taxiunternehmen benötigt ein Programm zur Erfassung und Abrechnung von Fahrten. Das Grundprogramm dafür ist in Labor 1 bis 3 erstellt worden; es soll in Labor 4 und 5 so erweitert werden, dass es zur Fahrtenplanung eingesetzt werden kann.

Fahrtenplanungsprobleme lassen sich gut mit Hilfe von Graphen modellieren und lösen. Als Vorbereitung auf die Modellierung von Graphen sind in Labor 4 zwei Klassen zu erstellen: „Node“ und „Edge“. Angewandt auf Fahrtenplanungsprobleme bilden Objekte der Klasse „Node“ (Knoten) Orte ab, d.h. Ausgangs- und Endpunkte von Fahrten und Zwischenstationen. Objekte der Klasse „Edge“ verbinden Knoten, und modellieren so Wege bzw. Fahrtstrecken.

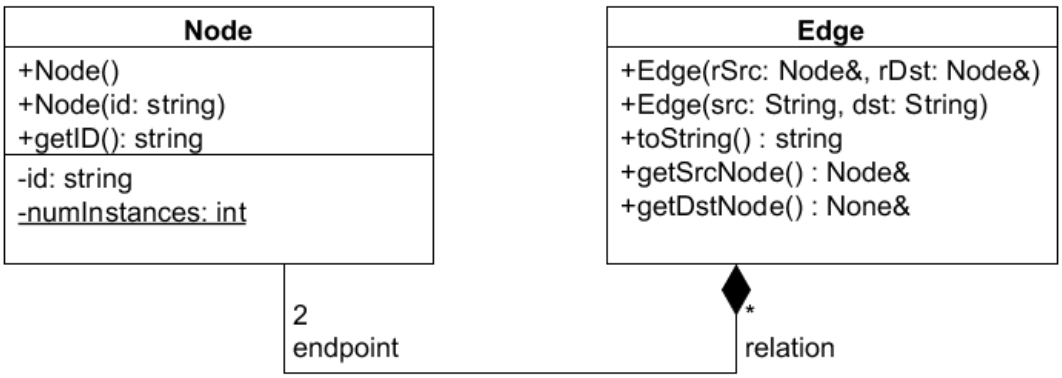
Aufgabenstellung

Aufgabe 1 - Klassen

Hinweise:

- Jede Klasse soll in einer separaten Header- und Quelldatei implementiert werden!
- Achten Sie generell darauf, dass die Member-Variablen `private` definiert werden!
- Benutzen Sie die C++ String-Klasse `std::string`, nicht die Strings, die Sie aus C kennen!

UML-Klassendiagramm



Klasse „Node“

a. Erstellen Sie 2 Konstruktoren:
   1. Bei der Instanziierung eines Objektes mit dem ersten Konstruktor soll ein Knoten-Bezeichner (`id`) angegeben werden können.
   2. Es soll zusätzlich ein Standardkonstruktor definiert werden. Dieser soll einen Bezeichner nach dem Schema: „Node_0001“, „Node_0002“ usw. generieren. Die Nummer ist die Anzahl aller bisher erzeugten Instanzen der Node-Klasse, damit die ID eindeutig ist. Dazu soll eine statische Klassenvariable `s_numInstances` zur Hilfe genommen werden, die bei jedem Konstruktoraufruf zu inkrementieren ist.

b. Erstellen Sie eine Member-Funktion `getId()`, die den Knotenbezeichner als String zurückgibt.

Klasse „Edge“

c. Definieren Sie die Klasse „Edge“ mit Referenzen auf zwei „Node“-Objekte als Member-Variablen: `m_srcNode` als Referenz auf den Startknoten, und `m_dstNode` als Referenz auf

den Zielknoten.
Hinweis: Die Klasse „Edge“ ist also eine gerichtete Kante.

d. Erstellen Sie zwei Konstruktoren:
   1. Der erste Konstruktor soll zwei Referenzen auf „Node“-Objekte als Parameter übernehmen.
   2. Der zweite Konstruktor soll zwei Strings als Parameter übernehmen. Er soll zwei „Node“-Objekte erzeugen und dabei die Strings als Knotenbezeichner verwenden.

e. Erstellen Sie folgende Member-Funktionen:
   - `toString()` soll einen `std::string` zurückgeben in der Form „`<SrcId> -> <DstId>`“, also beispielsweise „`Node_0001 -> Node_1234`“.
   - `getSrcNode()` und `getDstNode()` sollen jeweils eine Referenz auf den Start- bzw. den Zielknoten zurückgeben.

Aufgabe 2 – Exceptions, Try-Catch-Block

a. Stellen Sie im Konstruktor der „Node“-Klasse sicher, dass die Knotenbezeichner nur alphanumerische Zeichen enthalten (A-Z, a-z, 0-9). Werfen Sie andernfalls im Konstruktor eine Exception (als String-Typ) mit einer entsprechenden Fehlermeldung!
Hinweis: Sie können die Funktion `isalnum` aus der Standardbibliothek verwenden.

b. Implementieren Sie eine Hauptanwendung, in der Sie zwei Node-Objekte erzeugen. Testen Sie dabei die Exception, indem Sie einen Knotenbezeichner mit ungültigen Zeichen eingeben. Was passiert, wenn Sie die Exception nicht in einem `try-catch`-Block abfangen? Dokumentieren Sie die Ergebnisse als Kommentar im Quellcode!

c. Betten Sie die Instanziierung nun in einen `try-catch`-Block ein, um die Exception abzufangen und geben Sie eine Meldung aus, wenn eine Exception gefangen wurde.

Aufgabe 3 – Dynamische Speicherverwaltung

a. Legen Sie dynamisch (mit Hilfe des `new`-Operators) zwei weitere „Node“-Objekte im `try`-Block an. Initialisieren Sie dabei ein Objekt mit einem ungültigen Bezeichner. Wird dieses Objekt überhaupt erzeugt, wenn eine Exception im Konstruktor geworfen wird? Reservierter Speicher soll mit Hilfe von `delete` wieder freigegeben werden. Dokumentieren Sie die Ergebnisse!
Hinweis: Die Pointer müssen außerhalb des `try`-Blocks deklariert werden, damit Sie auch im `catch`-Block Zugriff darauf haben.

Aufgabe 4 – Eigene Exception-Klasse

a) Implementieren Sie eine Klasse „NodeIdException“ mit einer Funktion `getError()`, die die Fehlermeldung als `std::string` zurück gibt. Werfen Sie nun diese Klasse anstelle des Strings im Konstruktor der „Node“-Klasse als Exception. Passen Sie ihren `try-catch`-Block in der Hauptanwendung entsprechend an. Lassen Sie die vorherige Lösung bitte als Kommentar im Quellcode!

b) Verwenden Sie `const`-Referenzen im `catch`-Block. Worauf muss dabei in der Funktion `getError()` geachtet werden? Testen Sie, und dokumentieren Sie ihre Ergebnisse als Kommentar im Quellcode.

Aufgabe 5 – Exception über 2 Hierarchieebenen

a. Erzeugen Sie zwei „Edge“-Objekte. Verwenden Sie dabei die beiden unterschiedlichen Konstruktoren.

b. Testen Sie das Verhalten des Edge-Konstruktors, der zwei Id-Strings akzeptiert, indem Sie einen ungültigen Knotenbezeichner angeben. Überprüfen Sie, ob die Exception in Ihrer Hauptanwendung abgefangen werden kann und Dokumentieren Sie die Ergebnisse.