hochschule mannheim

Techniken der Programmentwicklung (TPE)
Sommersemester 2014

FAKULTÄT FÜR INFORMATIK

Pflichtübung 3

Ausgabe: 09.05.2014
Abgabe: 25.05.2014 (23:50 Uhr)
Testat (IMB): 26.05.2014 (10:30–12:45 Uhr)
Testat (UIB): 30.05.2014 (15:20–17:30 Uhr)

Aufgabe 1: Graph — 100 Punkte

Leider wurde Ihre Spiel Racewars kein großer Erfolg, da zum einen die Oberfläche wenig ansprechend und zum anderen die Menschen viel zu übermächtig waren. Deshalb wurden Sie entlassen und sind auf der verzweifelten Suche nach einer neuen Anstellung. An einem schneereichen Tag werden Sie in Moskau von einem Herrn im langen Trenchcoat angesprochen, der Ihnen eine lukrative Arbeit in den USA anbietet.

Ihr neuer Arbeitgeber ist eine Sicherheitsagentur aus den Vereinigten Staaten, die aus diversen Gründen ungenannt bleiben möchte. Das Ziel Ihrer Aufgabe besteht darin, in einem Geflecht von Beziehungen (einem gerichteten Graphen) nach bestimmten Eigenschaften zu suchen. Da man Ihnen nicht verraten möchte, welche Daten letztendlich in dem Graphen gespeichert werden, müssen Sie mit generischen Typen arbeiten. So hat die Agentur später die Möglichkeit, beliebige Daten in Ihrem Werkzeug zu speichern und zu suchen.

(a) **Node**

Ein Graph besteht aus Knoten (`Node`). Jeder Knoten hat einen Namen (`String`) und einen Wert beliebigen Typs – hier wird die Agentur später ihre eigenen Datentypen verwenden. Weiterhin hat der Knoten beliebig viele Kinder. Um den Knoten typsicher zu gestalten, soll er als generische Klasse implementiert werden, sodass der Typ des im Knoten gespeicherten Werts eindeutig festgelegt werden kann.

Jeder Knoten hat folgende Methoden:

- `addChild` – Hinzufügen eines Kindknotens
- `getChildren` – Auslesen aller Kindknoten
- `getName` – Auslesen des Namens des Knotens
- `getValue` – Auslesen des Wertes des Knotens

Implementieren Sie eine entsprechende *generische Klasse* `Node` mit den genannten Methoden. Für die Verwaltung der Kinder verwenden Sie bitte intern und auch in der Schnittstelle die weiter unten beschriebenen Listen.

(b) **Graph**

Der `Graph` besteht aus einer beliebigen Anzahl von Knoten, wobei einer der Knoten als Anfangsknoten ausgezeichnet wird. Von diesem Anfangsknoten aus können Sie alle anderen Knoten erreichen. Der Graph besitzt folgende Methoden:

- `search` – Suchen nach allen Knoten mit einem bestimmten *Wert*. Das Suchverfahren soll hierbei von Außen mitgegeben werden können (siehe unten).
- `copyInto` – Kopieren aller Knote in eine übergebene Liste.

(c) **Suchstrategie**

Für die Suche im Graphen schreiben Sie bitte ein Interface `SearchStrategy`. Dieses Interface hat zwei Methoden. Zum einen kann man, ausgehend von einem Startknoten, den Graphen nach Knoten mit einem bestimmten Wert durchsuchen und bekommt alle passenden Knoten zurück (`search`); zum anderen kann man den Weg den die Suche beim letzten Durchlauf durch den Graphen genommen hat auslesen (`getPath`).

Implementieren Sie danach das Interface für zwei unterschiedliche Suchstrategien:

- `Breitensuche` – eine Breitensuche
- `Tiefensuche` – eine Tiefensuche

Hinweis: Denken Sie daran, dass es sich um einen gerichteten Graphen handelt, der durchaus *Zyklen* enthalten kann. Sie müssen daher bei der Suche entsprechende Vorkehrungen treffen, um bei der Suche nicht in eine Endlosschleife bzw. unendliche Rekursion zu geraten.

(d) **Listen**

Für die Verwaltung von Listen, sowohl von Knoten, als auch von anderen Objekten, benötigen Sie entsprechende Klassen und Interfaces. Implementieren Sie daher in einem ersten Schritt das folgende Interface für die Verwaltung von Listen mit der Klasse `ListImpl`:

```
/**
 * Einfache Datenstruktur zur Verwaltung einer Reihe von Elementen.
 *
 * @param <T> Typ der gespeicherten Elemente.
 */
public interface List<T> extends Iterable<T> {

   /**
    * Überprüft, ob ein Element bereits vorhanden ist.
    *
    * @param e Element auf das geprüft werden soll
    * @return true wenn vorhanden, andernfalls false
    */
      public abstract boolean contains(Object e);

      /**
       * Fügt am Ende ein Element hinzu.
       *
       * @param e Element, das hinzugefügt werden soll.
       * @return ist immer true
       */
      public abstract boolean add(T e);

      /**
       * Fügt am Anfang ein Element hinzu.
       *
       * @param e Element, das angefügt werden soll.
       */
      public abstract void addFirst(T e);

      /**
       * Liefert das erste Element zurück, ohne es zu entfernen.
       *
       * @return das erste Element.
       */
      public abstract T peekFirst();

      /**
       * Überprüft, ob Elemente vorhanden sind.
       *
       * @return true wenn die Datenstruktur leer ist, andernlass false
       */
      public abstract boolean isEmpty();

      /**
       * Entfernt das erste Element und liefert es zurück.
       *
       * @return das erste Element
       */
      public abstract T pollFirst();

      /**
       * Löscht den Inhalt der List.
```

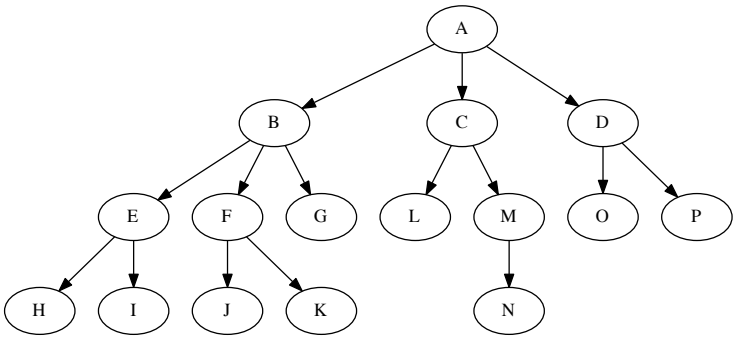
```
         */
        public abstract void clear();
}
```

Erstellen Sie nun eine spezialisierte Version des Interfaces (`NodeList`), die nur Knoten verwalten kann. Implementieren Sie auch diese mit der Klasse `NodeListImpl`.

Achtung: Sie können *erhebliche* Implementierungsaufwände sparen, wenn Sie eine geschickte Vererbungsbeziehung zu der Klasse `java.util.LinkedList` eingehen.

(e) **Test**

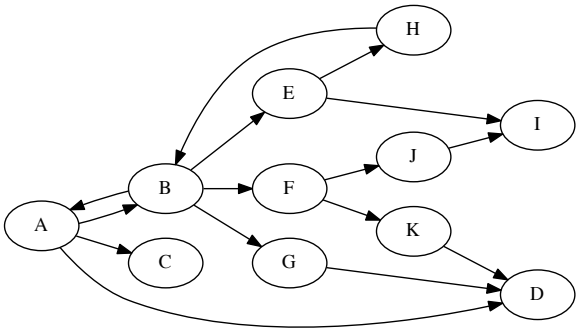
Testen Sie Ihre Implementierung mit Unit-Tests. Zeigen Sie auch, anhand des in der folgenden Abbildung dargestellten Graphen, dass Ihre Breiten- bzw. Tiefensuche die Knoten in der erwarteten Reihenfolge besuchen.



- Tiefensuche: [A, B, E, H, I, F, J, K, G, C, L, M, N, D, O, P]
- Breitensuche: [A, B, C, D, E, F, G, L, M, O, P, H, I, J, K, N]

(f) **Simulation**

Die folgende Abbildung zeigt einen beispielhaften Graphen.



Simulieren Sie diesen Graphen und gehen Sie davon aus, dass der im Knoten gespeicherte Wert seinem Namen entspricht. Suchen Sie dann den Knoten „K“ und geben Sie den Suchpfad sowohl für die Breiten- als auch die Tiefensuche aus.

Achtung
Bitte beachten Sie folgendes, damit es nicht zu unnötigen Punktabzügen in der Bewertung kommt:

- Benennen Sie Klassen und Methoden konsistent und verständlich. Klassen sollten Nomen als Namen, Methoden Verben haben.
- Dokumentieren Sie alle Klassen, Interfaces, Konstruktoren und Methoden mit JavaDoc. Dokumentieren Sie auch alle Parameter, Rückgabewerte und Ausnahmen.
- Formatieren Sie Ihren Code konsistent. Ein guter Standard sind 4 Spaces Einrückung pro Ebene ohne Verwendung von Tabulatoren.
- Halten Sie sich an die Sun Java-Code-Convention
  http://www.oracle.com/technetwork/java/codeconventions-150003.pdf
- Programmieren Sie defensiv und testen Sie Eingabewerte auf deren Gültigkeit. Ihr Programm darf auch mit Daten nicht abstürzen, die außerhalb der Aufgabenstellung liegen.
- Machen Sie keine Konsoleneingaben oder -Ausgaben, es sei denn die Aufgabe fordert dies explizit.
- Halten Sie Daten und Methoden zusammen. Trennen Sie diese nicht unnötig auf.
- Kopieren Sie keinen Code sondern versuchen Sie mit den bekannten Mitteln der Objektorientierung Code-Duplikate zu vermeiden.