Grundlagen der Informatik 1

Wintersemester 2009/2010

Prof. Dr. Max Mühlhäuser, Dr. Rößling
http://proffs.tk.informatik.tu-darmstadt.de/teaching

Übung 12 Version: 1.1 25.01.2010

---

1 Mini Quiz

- ☐ Die Reihenfolge der catch-Blöcke ist nicht von Bedeutung.
- ☐ Ein Compiler kann alle Arten von Fehlern (bis auf Laufzeitfehler) entdecken und signalisieren.
- ☐ Die beim Ausführen einer Methode auftretenden Ausnahmen können im catch Block behandelt werden.
- ☐ Während der Laufzeit des Programms auftretende Fehler müssen behandelt werden.
- ☐ Ausnahmen müssen unmittelbar in der Methode behandelt werden, in der sie auftreten.
- ☐ Man kann leicht nachweisen, dass ein Programm fehlerfrei ist.

2 Verständnis

1. Welche Gemeinsamkeiten besitzen die Ausnahmeklassen Exception und Error? Was unterscheidet sie? Welche von beiden sollte man abfangen?
2. Erklären Sie, was Syntax-, Laufzeit-, Intentions- und lexikalische Fehler sind. Welche dieser Fehler werden vom Compiler erkannt?
3. Wie hängen die Begriffe Throwable, throws und throw zusammen? Wo sollte man was verwenden?
4. Ist es sinnvoll, eigene Ausnahmeklassen zu definieren? Welche Vorteile ergeben sich hieraus?

3 Fehlersuche

1. Betrachten Sie den folgenden Java-Code. Beheben Sie alle syntaktischen Fehler.

```
1  public static int[] reverseArray (int[] source) throw Exception {
2    int length = source.length();
3    int[] inverted;
4    int i=0;
5
6    try {
7      inverted = new int[length] ;
8
9      while (i < length){
10       inverted[i] = source[i];
```

```
11          i++;
12        }
13      }
14      catch (Exception) {
15        System.out.println("Caught Exception...");
16      }
17      catch (IndexOutOfBoundsException e) {
18        System.out.println("Caught Exception: " + e);
19      }
20      final {
21        inverted = new int[length()];
22        inverted = source ;
23      }
24      return inverted;
25  }
```

2. Was passiert generell beim Aufruf der Funktion *reverseArray*?

   Die Funktion wird von Java nicht kompiliert - auch **ohne** vorhandene syntaktische Fehler, weswegen? Wie können Sie die Funktion dennoch kompilieren, was müsste man dazu beheben?

   Hinweis: Betrachten Sie bitte nochmal die Mini Quiz Fragen...

3. Suchen und beheben Sie die vorhandenen Intentionsfehler (Fehlerquelle 'Mensch').

4 Funktionen höherer Ordnung in Java

Erinnern Sie sich noch an die Funktion fold aus dem Scheme-Teil der Vorlesung? Zur Erinnerung, der Vertrag lautet:

```
1  ;; fold : (X Y -> Y) Y -> ((list of X) -> Y)
```

Machen Sie sich nochmal kurz klar, welche Aufgabe die folgende Variante von fold erfüllt.

```
1  (define (fold f n)
2   (lambda (x)
3     (if (empty? x)
4       n
5       (f (first x) ((fold f n) (rest x ))))))
```

Wir wollen nun eine ähnliche Funktion in Java implementieren. Zunächst stellen wir fest, dass eine direkte Übersetzung nach Java nicht möglich ist, da Java keine Funktionen höherer Ordnung unterstützt. Mit Hilfe eines Interfaces kann jedoch ein solches Verhalten simuliert werden.

1. Deklarieren Sie ein Interface namens BinaryOp, das eine binäre Operation `apply` über ganzen Zahlen deklariert. Beide Argumente sowie der Rückgabewert sollen vom Typ int sein.

2. Schreiben Sie jetzt die Klassen Adder und Multiplier, die das Interface BinaryOp implementieren und in der Methode `apply` die Addition bzw. Multiplikation zur Verfügung stellen.

3. Implementieren Sie nun eine Methode mit folgender Signatur: **int** jfold(**int**[] a, BinaryOp op, **int** s)

   Diese Methode soll analog zu fold die ihr übergebene Operation *op* schrittweise auf alle Elemente des Arrays *a* anwenden. Dabei entspricht *s* dem Argument *n* in fold.

4. Erklären Sie nun, worin sich jfold und fold unterscheiden.

5 Polymorphie

Betrachten Sie folgenden Java Code:

```
1  interface MyInterface {
2    public int g();
3  }
4
5  class Base implements MyInterface {
6    public int i = 1;
7    public int g() { return f(); }
8    protected int f() { return i; }
9  }
10
11 class Derived extends Base {
12   public int i = 2;
13   protected int f() { return -i; }
14   protected int h() { return i + i; }
15   protected int p() { return i * i; }
16 }
17
18 public class Client {
19   public static void main (String args[]){
20     Base base = new Base() ;
21     Derived derived = new Derived();
22     MyInterface restricted = derived;
23
24     int temp = base.i;      // a)
25     temp = base.g();        // b)
26     base = derived;
27     temp = base.i;          // c)
28     temp = base.g() ;       // d)
29     temp = restricted.g(); // e)
30     restricted = base;
31     temp = restricted.g(); // f)
32     System.out.println(temp);
33   }
34 }
```

1. Welchen statischen und dynamischen Typ hat die Variable base zum Zeitpunkt a) bzw. c)?
2. Welchen Wert hat die Variable **temp** zu den Zeitpunkten a), b), c), d) e) und f)?
3. Betrachten Sie nun die folgenden Aufrufe:

```
1  Base base = new Base();
2  Derived derived = new Derived();
3  int temp = base.g();
4  derived = base;
5  base = new Derived();
6  MyInterface t = new Derived();
7  t = base;
8  temp = t.f();
9  derived = t;
10 base = t;
11 t = derived;
12 temp = t.p();
13 temp = t.g();
14 temp = derived.p();
```

   Welche Aufrufe sind erlaubt, welche werfen einen Fehler? Treten die Fehler zur Laufzeit oder bereits beim Kompilieren auf? Es gibt keine Folgefehler - kommentieren Sie verbotene Aufrufe aus und ignorieren Sie diese bei der weiteren Fehlersuche.

Hausübung

Die Vorlagen für die Bearbeitung werden im GdI1-Portal bereitgestellt. Kommentieren Sie Ihren selbst erstellten Code. Die Hausübung muss bis zum Abgabedatum im GdI1-Portal abgegeben werden. Der Fachbereich Informatik misst der Einhaltung der Grundregeln der wissenschaftlichen Ethik großen Wert bei. Zu diesen gehört auch die strikte Verfolgung von Plagiarismus. Mit der Abgabe Ihrer Hausübung bestätigen Sie, dass Sie bzw. Ihre Gruppe alleiniger Autor des gesamten Materials sind. Falls Ihnen die Verwendung von Fremdmaterial gestattet war, so müssen Sie dessen Quellen deutlich zitieren. Falls Sie die Hausübung in einer Lerngruppe bearbeitet haben, geben Sie dies bitte deutlich bei der Abgabe an. Alle anderen Mitglieder der Lerngruppe müssen als Abgabe einen Verweis auf die gemeinsame Bearbeitung einreichen, damit die Abgabe im Portal auch für sie bewertet werden kann.
Abgabe: Bis spätestens Freitag, 05.02.2010, 16:00 Uhr.

6 Planung einer Eisenbahnstrecke (12 Punkte)

Im Kolonialzeitalter kam der Eisenbahn eine wesentliche Rolle bei der Erschließung neuen Territoriums zu. Durch neue Eisenbahnstrecken wurden enorme Distanzen überbrückt. Ein Paradebeispiel für eine solche (unvollendete) Eisenbahnstrecke ist die Kap-Kairo-Bahn, welche die britischen Kolonien in Afrika miteinander verbinden sollte. Aufgrund mangelnder finanzieller Mittel wurde diese Bahnstrecke niemals vollendet.
In jüngster Zeit gibt es wieder Überlegungen, diese Bahn zu realisieren bzw. die Lücken zwischen den bestehenden Teiltrassen zu schließen. Allerdings ist das Gelände in Afrika nicht immer bahntauglich, so dass Sie beauftragt wurden, zunächst die generelle Machbarkeit der einzelnen Abschnitte zu untersuchen. Dazu stehen Ihnen Satellitendaten der jeweiligen Regionen zur Verfügung.

6.1 Grundstruktur (0.5 Punkte)

Erstellen Sie zunächst die Klasse `TrainSimulation`. Diese soll als interne Datenstruktur area ein `char[][]` verwenden; diese Datenstruktur repräsentiert eine Karte des zu untersuchenden Gebiets. Eine korrekte Karte ist ein rechteckiges Gebiet, wobei jeder Teil durch eines von fünf Symbolen repräsentiert wird:

- '#' für eine größere Stadt,
- '0' für ein Stück hinreichend ebenes Land,
- '~' für ein Sumpfgebiet, das aus finanziellen Gründen nicht trockengelegt werden kann,
- sowie '^' für ein Gebirge.

Implementieren Sie die folgenden Methoden der Klasse `TrainSimulation`:

- Konstruktor TrainSimulation(**char** [][] area)
- Getter- und Setter-Methoden für die interne area Datenstruktur
- Eine Methode toString(), die eine Stringrepräsentation der Area ausgibt. Eine solche Repräsentation sollte folgendermaßen aussehen (die Leerzeichen dienen der besseren Lesbarkeit und sollen in Ihrer Ausgabe nicht auftreten):

```
1 0 0 0 0 0 ~ ~ 0 0 #
2 # 0 0 0 ~ ~ 0 0 0 0
3 0 0 ^ ^ ^ ^ # 0 0 0
4 ^ 0 0 0 0 0 0 0 0 0
5 0 0 ~ ~ ~ ~ ^ ^ ^ ^
```

6.2 Gültige Karten (2 Punkte)

Damit die Untersuchung einer Karte lohnt, muss es *mindestens zwei Städte*, *ein Ebenenfeld* und *ein Geländehindernis* geben—sonst braucht man keine Untersuchung. Schreiben Sie dazu die Methoden **boolean** hasTwoCities(), **boolean** hasPlainsElement() und **boolean** hasTerrainObstacle().
Sobald eine neue Area angegeben wird—im Konstruktor oder durch setArea()—soll geprüft werden, ob diese Karte gültig ist und ansonsten eine entsprechende Exception geworfen werden. Schreiben Sie dafür die Exceptions NotEnoughCitiesException, NoPlainsException bzw. NoObstacleException. Diese Exceptions sollen von der Exception InvalidAreaException erben.

6.3 Erreichbarkeitsanalyse (3 Punkte)

Implementieren Sie die Methode **void** reachable(**int** x, **int** y), welche die Koordinaten einer Stadt entgegennimmt und berechnet, welche Gebiete von dieser Stadt aus überhaupt mit einer Eisenbahnlinie erreicht werden können. Die Koordinaten werden dabei direkt zur Indizierung des Feldes genutzt, wobei die x-Koordinate an der ersten Stelle steht[1]. Mit den Koordinaten (x,y) ist daher das Element area[x][y] gemeint.
Eine Eisenbahnlinie kann nur auf Ebenenfeldern gebaut werden und soll durch das Symbol '=' repräsentiert werden. Nehmen Sie an, dass Stadfelder bereits eine Eisenbahn haben. Eine Eisenbahnstrecke kann durch ein Stadtfeld hindurch führen, ersetzt es aber nicht. Eisenbahnstrecken können nur auf Feldern gebaut werden, welche vertikal oder horizontal an ein bereits angeschlossenes Feld angrenzen. Diagonale Verbindungen sind nicht möglich.
Hinweis: Sie sollten sich beim Lösen am Floodfill-Algorithmus (siehe `http://de.wikipedia.org/wiki/Floodfill`) orientieren. Sie dürfen das Ausgangsarray dabei verändern.
Hinweis: Beachten Sie bitte, dass wir in x- und y-Richtung bei **0** mit dem Zählen beginnen. Außerdem betrachten wir erst die x-Koordinate (also die Zeile der Matrix), dann die y-Koordinate. Der Punkt (1,0) ist daher der 2. Eintrag (x=1) in der ersten Zeile (y=0).
Beispiel: reachable(1,0)

```
0 | 0 # ^ 0 0 0 0          = # ^ 0 0 0 0
1 | 0 0 ^ ^ ^ 0 0          = = ^ ^ ^ 0 0
2 | 0 0 ^ 0 # ~ 0    —>    = = ^ = # ^ 0
3 | 0 0 # 0 0 ~ 0          = = # = = ~ 0
4 | ~ ~ ~ ~ ~ ~ ~          ~ ~ ~ ~ ~ ~ ~
5 | 0 0 0 0 0 0 0          0 0 0 0 0 0 0
```

reachable soll zwei verschiedene Exceptions auslösen können:

- InvalidCoordinatesException , wenn die angegebenen Koordinaten nicht gültig sind, d.h. außerhalb der Karte liegen.
- NotACityException, wenn die angegebenen Koordinaten nicht auf eine Stadt zeigen.

Hinweis: Falls auf der Karte zwei Städte direkt nebeneinander liegen, funktioniert der Algorithmus nicht ohne größeren Aufwand. Sie brauchen dies nicht zu berücksichtigen.

6.4 Kostenanalyse (4.5 Punkte)

Nachdem nun überhaupt mögliche Strecken ermittelt wurden, wollen wir nun die wahrscheinlichen Kosten einer Strecke ermitteln. Das zugrundeliegende Modell ist hier etwas vereinfacht: Eine Geländefeld mit einer Eisenbahnstrecke zu bebauen kostet eine Einheit einer hier nicht weiter betrachteten Währung. Soll eine Eisenbahnstrecke durch eine Stadt führen, muss auch hier genau eine Einheit investiert werden, um baulichen Veränderungen Rechnung zu tragen (und Ihnen das Leben nicht allzu schwer zu machen).

[1] im Gegensatz zur üblichen zeilenbasierten Speicherung des Arrays

Schreiben sie eine Methode **int** [][] analyzeCosts(**int** x, **int** y), welche eine angepasste Version des Algorithmus aus dem vorigen Teil verwendet. Die Parameter sollen hierbei wieder auf eine Stadt zeigen. Ist dies nicht der Fall, signalisieren Sie dies durch die passende Exception.
Der Rückgabewert der Methode soll ein zweidimensionales Array sein, welches für jedes Feld die zu erwartenden Kosten enthält, die anfallen, wollte man es durch eine Eisenbahnlinie an die angegebene Stadt anbinden. Dabei soll ein unerreichbares Feld die Kosten -1 haben, das Ausgangsfeld die Kosten 0 und alle anderen Felder die minimalen Kosten der Nachbarfelder plus eins.
Beispiel: analyzeCosts(1,0)

```
1 | 0 # ^ 0 0 0 0          1  0  −1 −1 −1 −1 −1
2 | 0 0 ^ ^ ^ 0 0          2  1  −1 −1 −1 −1 −1
3 | 0 0 ^ 0 # ~ 0   —>     3  2  −1  6  7 −1 −1
4 | 0 0 # 0 0 ~ 0          4  3   4  5  6 −1 −1
5 | ~ ~ ~ ~ ~ ~ ~         −1 −1  −1 −1 −1 −1 −1
6 | 0 0 0 0 0 0 0         −1 −1  −1 −1 −1 −1 −1
```

Hinweise:

- Sie werden ein weiteres privates Attribut benötigen. Sie brauchen dafür keine Getter und Setter zu schreiben.
- Überlegen Sie sich bei Ihrer Lösung zunächst, wie sie vorgehen müssen. Was passiert, wenn das Nachbarfeld schon zugewiesene Kosten hat? Wann müssen Sie diese Kosten ändern, und wann können Sie einen rekursiven Aufruf beenden?

6.5 Benutzerinteraktion (2 Punkte)

Schreiben Sie eine Klasse TrainDialog (das Gerüst wird im Portal bereitgestellt), die den Benutzer mit der TrainSimulation interagieren lässt. Die Klasse soll die Area mit vorgegebenen Werten initialisieren und den Benutzer über eine InvalidAreaException mit einer passenden Fehlermeldung informieren. Danach soll die Klasse das aktuelle Gebiet anzeigen, vom Benutzer die x- und y-Koordinaten einer Ausgangsstadt erfragen und dann die erreichbaren Felder ermitteln.
Sollte die Eingabe der Koordinaten nicht korrekt sein, soll der Benutzer eine passende Fehlermeldung erhalten. Die Darstellung der Kosten wird hier nicht gefordert. Bei Eingabe von -1 soll das Programm beendet werden.