Übungen zu EIDI

Version 1.1.0

Johannes Stöhr

8. April 2019

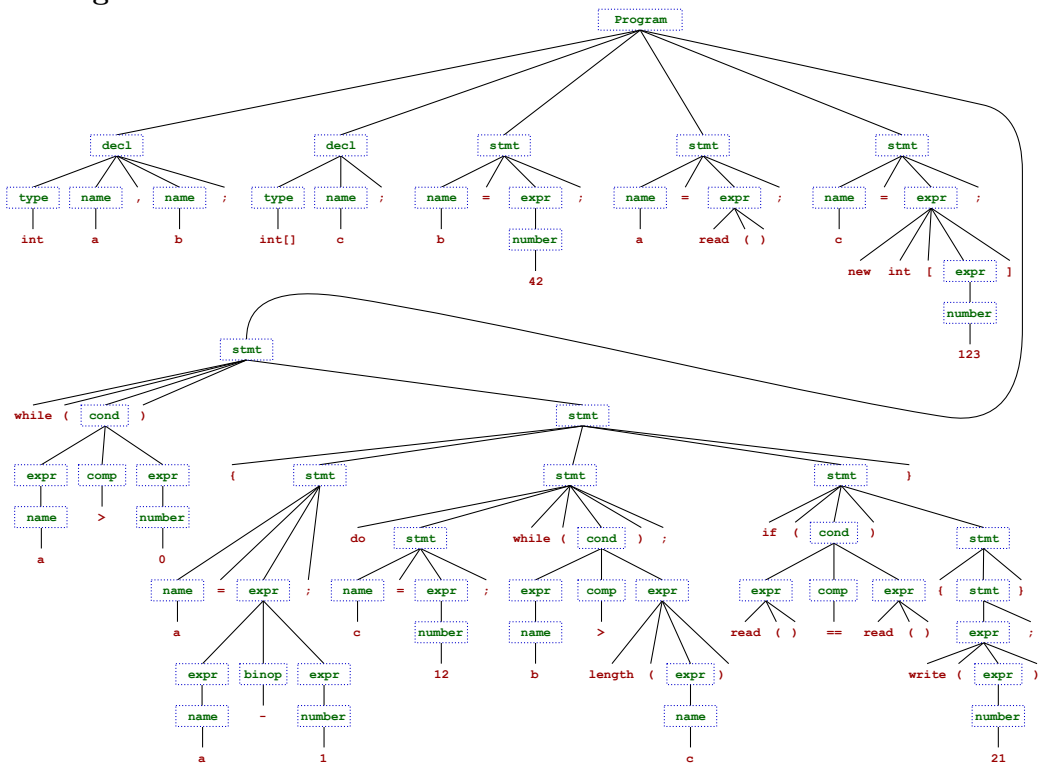
ACHTUNG: LÖSUNGEN

1. **Syntaxbäumchen**

(a) Zeichnen sie ein Syntaxbaum zu folgendem MiniJavaProgramm (mit der Grammatik von PGdP-Blatt 14):

```
1  int a, b;
2  int[] c;
3  b = 42;
4  a = read();
5  c = new int[123];
6  while(a > 0) {
7      a = a - 1;
8      do {
9          c = 12;
10     } while(b > length(c));
11     if(read() == read()) {
12         write(21);
13     }
14 }
```

Lösung:



(b) Würde dieses Programm auf hier fehlt noch was kompilieren? Warum (nicht)?

Lösung: Kompiliert, weil es einen gültigen Syntaxbaum gibt und auch der Kontext der Grammatik stimmt, d.h. bei Mini-Java: alle Variablen wurden vorher deklariert. Eine Typprüfung nimmt Mini-Java nicht vor, daher ist das Zuweisen eines `int`s zu einem Array wie in Zeile 9 (leider) erlaubt. Die Begründung, dass ein anständiger Compiler so eine Zuweisung nie erlauben dürfte, wäre jedoch auch richtig.

2. **Vervollständigen Sie den Lückentext:**

```
public class Randomsort {
    public static void sort(int[] array) {
        while(!isSorted(array)) {
            int index = ( int ) (Math.random() * (array.length - 1)) + 1 ;
            // 0.0 <= Math.random() < 1.0
            int remember = ______array[index]______;
            array[____index____] = array[0];
            _______array[0] = remember_______;
        }
    }

    ___private static boolean___ isSorted(int[] array) {
        for(___int i = 1; i < array.length; i++___) {
            if(array[i] < array[i - 1]) {
                return false;
            }
        }
        ______return true______;
    }
}
```

3. **Polymorphie**

Was wird bei den folgenden Aufrufen ausgegeben? Betrachten sie jede Ausgabe getrennt von den anderen. Geben Sie an, sofern eine Ausgabe oder Objekterzeugung (`Obj.Erzeugung`) nicht kompiliert. Sollte eine Exception geworfen werden, soll angegeben werden um welche es sich handelt.

```
public class Polymorphie {
    public static void main(String[] args) {
        int intOne = 1;
        int intTwo = 3;
        double doubleOne = 3;
        double doubleTwo = 7.0;
        System.out.println(foo(intOne));              // Ausgabe  1
        System.out.println(foo(doubleTwo));           // Ausgabe  2
        System.out.println(foo(intTwo, intOne));      // Ausgabe  3
        System.out.println(foo(doubleOne, intTwo));   // Ausgabe  4
        System.out.println(foo(intOne, doubleOne));   // Ausgabe  5
        A a = new A();                                // Obj.Erzeugung 1
        B b1 = new B();                               // Obj.Erzeugung 2
        B b2 = new B(b1);                             // Obj.Erzeugung 3
        C c1 = new C();                               // Obj.Erzeugung 4
        C c2 = new C(b2);                             // Obj.Erzeugung 5
        // Sehr viele; nicht unbedingt alle für die Übung,
        // manche auch für Zuhause, wenn man nochmal üben will
        System.out.println(a.goo(a));                 // Ausgabe  6
        System.out.println(a.goo(b2));                // Ausgabe  7
        System.out.println(a.goo(c2));                // Ausgabe  8
        System.out.println(a.goo(new E()));           // Ausgabe  9
        System.out.println(b1.goo(b1));               // Ausgabe 10
```

```
24          System.out.println(b2.goo(a));                  // Ausgabe 11
25          System.out.println(b1.equals(b2.goo(b2)));      // Ausgabe 12
26          System.out.println(c2.goo(a, b2));              // Ausgabe 13
27          System.out.println(c2.goo(b1, a));              // Ausgabe 14
28          System.out.println(c2.hoo());                   // Ausgabe 15
29          b1 = new C(b2);                                 // Obj.Erzeugung 6
30          System.out.println(b1.goo(b1));                 // Ausgabe 16
31          System.out.println(b1.goo(b2));                 // Ausgabe 17
32      }
33
34      private static double foo(double a, double b) {
35          return a > b ? a : b;
36      }
37
38      private static int foo(double a, int b) {
39          return foo((int) a);
40      }
41
42      private static int foo(int a, double b) {
43          return (int) foo((double) a, b);
44      }
45
46      private static int foo(int a) {
47          return 4 * a + 2;
48      }
49
50      public static class A {
51          public String toString() {
52              return "A";
53          }
54
55          public A goo(A a) {
56              return new A();
57          }
58
59          public C goo(D d) {
60              return (C) d;
61          }
62      }
63
64      public static class B extends A {
65          public B b;
66
67          public B(B b) {
68              this.b = b;
69          }
70
71          public B() {
72              b = null;
73          }
74
```

```
75          public String toString() {
76              return "B";
77          }
78
79          public A goo(B b) {
80              return this.b;
81          }
82
83          public B goo(A a, B b) {
84              return (B) b.goo(a);
85          }
86      }
87
88      public static class C extends B implements D {
89          public C(B b) {
90              super(b);
91          }
92
93          public String toString() {
94              return "C";
95          }
96
97          public A goo(A a) {
98              return a.goo(a);
99          }
100
101         public A goo(B b) {
102             return new B(null);
103         }
104
105         public A goo(B b, A a) {
106             return a.goo(b);
107         }
108
109         public B hoo() {
110             return new B(this.b);
111         }
112     }
113
114     public interface D {
115         public B hoo();
116     }
117
118     public static class E implements D {
119         public B hoo() {
120             return new C(null);
121         }
122     }
123 }
```

Lösung: Text in der Konsole oder Fehlergründe je Ausgabe:

1. `6`
2. Kompiliert nicht, es gibt keine Methode `Polymorphie.foo(double)` oder eine kompatible Überladung.
3. Kompiliert nicht, beide Methoden `Polymorphie.foo(int, double)` und `Polymorphie.foo(double, int)` kommen in Frage und keine ist näher an `foo(int, int)` als die jeweils andere. (mehrdeutiger Methodenaufruf)
4. `14`
5. `3`
6. `A`
7. `A`
8. Kompiliert nicht, da `A.goo(A)` und `A.goo(D)` beide ein `C`-Objekt entgegennehmen können, weil `C` Subtyp von `A` ist und `D` implementiert. Wie bei 3. ein mehrdeutiger Methodenaufruf.
9. Wirft zur Laufzeit eine `ClassCastException`, weil `E` zwar auch `D` implementiert, aber keine Unterklasse von `C` ist. Das neu erzeugte `E`-Objekt kann somit nicht zu `C` gecastet werden.
10. `null`
11. `A`
12. `true`
13. Es wird eine `ClassCastException` geworfen, weil das in `A.goo(A)` erzeugte `A`-Objekt in der Methode `B.goo(A, B)` zu `B` gecastet werden kann. `B` ist Unterklasse von `A` , aber nicht umgekehrt.
14. `A`
15. `B`
16. `B`
17. `B`

Zudem kompiliert Objekterzeugung 4 nicht, weil es keinen parameterlosen Konstruktor in der Klasse `C` gibt.

4. **Pingus Space Odyssey**

Pingu ist im Weltall unterwegs. Allerdings wurde sein Raumschiff von einem Weltraumeisbären angegriffen, weshalb er nun zum Flottenstützpunkt muss, um sich auf einen Gegenschlag vorzubereiten.

Dafür muss er sich durch den Weltraum bewegen, in dem es mehrere Beacons gibt. Jeder dieser Punkte ist mit einem oder mehreren anderen Punkten verbunden. Dies wird durch eine Adjazenzmatrix dargestellt. Das ist ein zweidimensionales int-Feld, in dem jeweils eine Verbindung von einem Punkt (Zeilenindex) zu einem anderen Punkt (Spaltenindex) an der jeweiligen Stelle durch eine „1“ gekennzeichnet ist. Es sind auch Schleifen und „Einbahnstraßen“ möglich! Jedoch gibt es auch Wurmlöcher als Verbindung, die mit einer „2“

gekennzeichnet sind. Diese erzeugen eine Kopie von Pingu (neuer Thread) in einem Paralleluniversum an dem Zielpunkt des Wurmlochs, der normale Pingu bleibt an seinem Punkt und kann sich frei einen anderen Weg aussuchen.

Jedoch müssen die Grenzen der Paralleluniversen gewahrt werden, daher dürfen zwei Kopien von Pingu nicht am selben Beacon existieren, da sonst die beiden Realitäten kollidieren und alles auseinandergerissen wird.

Damit Pingu nicht im Kreis fliegt, sollte sich jede Kopie von ihm merken, wo diese schonmal war. Wird Pingu kopiert weiß die neue Kopie nichts vom bisherigen Weg. Nur für die Bonusaufgabe darf der bisherige Weg übergeben werden. Sollte Pingu in einer Sackgasse landen, darf er einen Schritt zurückgehen und von dort einen anderen Weg wählen. Sollten alle Wege von einem Punkt nicht zum Ziel führen, darf er noch einen Schritt weiter zurückgehen und so weiter.

Sobald eine Kopie von Pingu das Ziel erreicht hat, können alle Pingus aufhören nach einem Weg zu suchen, da die Sternenflotte ein spezielles Funkgerät besitzt, mit dem sie auch mit Paralleluniversen kommunizieren kann. Daher sind dann automatisch alle Pingukopien gerettet.

Erweitern sie hierzu die Implementierung der Klasse `PenguinAstronaut`.

Bonus: Ein Erzähler schildert nach Pingus Ankunft die aufregende Reise durch das Weltall.

```
1  public class Space {
2      // 0 = keine Verbindung; 1 = Verbindung; 2 = Wurmloch
3      private int[][] adjacencyMatrix;
4
5      public int[][] getAdjacencyMatrix() {
6          return adjacencyMatrix;
7      }
8
9      public Space(int[][] adjacencyMatrix) {
10         this.adjacencyMatrix = adjacencyMatrix;
11     }
12
13     public static void main(String[] args) {
14         //Nur ein Beispiel
15         int[][] matrix = { { 0, 0, 2 }, { 0, 0, 0 }, { 0, 1, 1 } };
16         Space space = new Space(matrix);
17         PenguinAstronaut pingu = new PenguinAstronaut(space, 0, 2);
18         pingu.start();
19     }
20 }
21
22 public class PenguinAstronaut extends Thread {
23     public PenguinAstronaut(Space s, int from, int to) {
24         // "Standardkonstruktor" für main() in Space
25     }
26 }
```

Mögliche Lösung:

```java
import java.util.ArrayList;
import java.util.List;

public class PenguinAstronaut extends Thread {
    private static Object[] locks;

    private static synchronized void initLocks(int size) {
        // Initialise lock objects only once
        if (locks == null) {
            locks = new Object[size];
            for (int i = 0; i < locks.length; i++) {
                locks[i] = new Object();
            }
        }
    }

    private Space space;
    private int sizeOfSpace;
    private int start;
    private int to;
    private boolean[] visited;

    // For stopping all Threads at the end
    private PenguinAstronaut parentThread;
    private List<PenguinAstronaut> childThreads;

    // For telling the way afterward
    private List<Integer> visitedLocations;

    // Standard constructor for first Pingu
    public PenguinAstronaut(Space space, int start, int to) {
        this(space, start, to, null, new ArrayList<>());
    }

    // Constructor for every other Pingu
    public PenguinAstronaut(Space space, int start, int to,
        ↪  PenguinAstronaut parentThread, List<Integer> visitedLocs) {
        this.space = space;
        sizeOfSpace = space.getAdjacencyMatrix().length;
        this.start = start;
        this.to = to;
        visited = new boolean[sizeOfSpace];

        this.parentThread = parentThread;
        childThreads = new ArrayList<>();

        this.visitedLocations = visitedLocs;

        initLocks(sizeOfSpace);
    }
```

```
50
51      @Override
52      public void run() {
53          visit(start);
54      }
55
56      private void visit(int index) {
57          // Way was already found
58          if (this.isInterrupted())
59              return;
60
61          // Needs to be declared before synchronized
62          int nextVisit = 0;
63          // Lock current location
64          synchronized (locks[index]) {
65              // Found location of starfleet
66              if (index == to) {
67                  tellStory();
68                  stopSearching(this);
69                  return;
70              }
71              visitedLocations.add(index);
72              visited[index] = true;
73
74              /*
75               * New Pingus through wormholes
76               *
77               * matrix[index][i] = 1 means there is a way from
78               * the current location to the beacon with the index i
79               */
80              for (int i = 0; i < sizeOfSpace; i++) {
81                  if (space.getAdjacencyMatrix()[index][i] == 2 &&
                        ↪  !visited[i]) {
82                      PenguinAstronaut p = new PenguinAstronaut(space, i,
                            ↪  to, this, new ArrayList<>(visitedLocations));
83                      childThreads.add(p);
84                      p.start();
85                  }
86              }
87              // Visit next beacons
88              nextVisit = nextVisit(nextVisit, index);
89          }
90
91          while (nextVisit != -1) {
92              visit(nextVisit);
93              // We're back from a dead end
94              synchronized (locks[index]) {
95                  visitedLocations.remove(visitedLocations.size() - 1);
96                  // Determine if and where to go next
97                  nextVisit = nextVisit(nextVisit + 1, index);
98                  // Possibly going back one step, unlock current beacon
```

```
            }
        }
    }

    private int nextVisit(int seachStart, int index) {
        // Looking for the next Place to visit
        for (int i = seachStart; i < sizeOfSpace; i++) {
            if (space.getAdjacencyMatrix()[index][i] == 1 && !visited[i])
                return i;
        }
        return -1;
    }

    private void stopSearching(PenguinAstronaut stopping) {
        // Stop parent if it's not the one stopping
        if (parentThread != null && parentThread != stopping) {
            parentThread.stopSearching(this);
        }
        // Stop every child that is not the one stopping
        for (PenguinAstronaut p : childThreads) {
            if (p != stopping) {
                p.stopSearching(this);
            }
        }
        // Interrupt this Pingu's search
        this.interrupt();
    }

    private void tellStory() {
        for (Integer i : visitedLocations) {
            System.out.println("I was at beacon " + i);
        }
    }
}
```