

Übungen zu EIDI

Version 1.0.0

Johannes Stöhr

7. 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     }
```

- (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 **ints** zu einem Array wie in Zeile 9 (leider) erlaubt.

2. Vervollständigen Sie den Lückentext:

```
1 public class Randomsort {
2     public static void sort(int[] array) {
3         while(!isSorted(array)) {
4             int index = (int) (Math.random() * (array.length)) + 0;
5             // 0.0 <= Math.random() < 1.0
6             int remember = array[index];
7             array[index] = array[0];
8             array[0] = remember;
9         }
10    }
11
12    private static boolean isSorted(int[] array) {
13        for(int i = 1; i < array.length; i++) {
14            if(array[i] < array[i - 1]) {
15                return false;
16            }
17        }
18        return true;
19    }
20 }
```

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.

```
1 public class Polymorphie {
2     public static void main(String[] args) {
3         int intOne = 1;
4         int intTwo = 3;
5         double doubleOne = 3;
6         double doubleTwo = 7.0;
7         System.out.println(foo(intOne));           // Ausgabe 1
8         System.out.println(foo(doubleTwo));        // Ausgabe 2
9         System.out.println(foo(intTwo, intOne));    // Ausgabe 3
10        System.out.println(foo(doubleOne, intTwo)); // Ausgabe 4
11        System.out.println(foo(intOne, doubleOne)); // Ausgabe 5
12        A a = new A();                               // Obj.Erzeugung 1
13        B b1 = new B();                               // Obj.Erzeugung 2
14        B b2 = new B(b1);                             // Obj.Erzeugung 3
15        C c1 = new C();                               // Obj.Erzeugung 4
16        C c2 = new C(b2);                             // Obj.Erzeugung 5
17        // Sehr viele; nicht unbedingt alle für die Übung,
18        // manche auch für Zuhause, wenn man nochmal üben will
19        System.out.println(a.goo(a));                // Ausgabe 6
20        System.out.println(a.goo(b2));                // Ausgabe 7
21        System.out.println(a.goo(c2));                // Ausgabe 8
22        System.out.println(a.goo(new E()));           // Ausgabe 9
23        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, wissen beide Kopien, welche Punkte der bisherige Weg beinhaltet hat. 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 Pingu kopien gerettet.

Erweitern sie hierzu die Implementierung der Klasse `PenguinAstronaut`.

Bonus: Pingu schildert auf dem Flottenstützpunkt seinen gefährlichen Weg durch das Universum.

```
1 public class Space
2 {
3     // 0 = keine Verbindung; 1 = Verbindung; 2 = Wurmloch
4     private int[] [] adjacencyMatrix;
5
6     public int[] [] getAdjacencyMatrix() {
7         return adjacencyMatrix;
8     }
9
10    public Space(int[] [] adjacencyMatrix) {
11        this.adjacencyMatrix = adjacencyMatrix;
12    }
13
14    public static void main(String[] args) {
15        //Nur ein Beispiel
16        int[] [] matrix = { { 0, 0, 2 }, { 0, 0, 0 }, { 0, 1, 1 } };
17        Space space = new Space(matrix);
18        PenguinAstronaut pingu = new PenguinAstronaut(space, 0, 2);
19        pingu.start();
20    }
21 }
22
23 public class PenguinAstronaut extends Thread {
24     public PenguinAstronaut(Space s, int from, int to) {
25         // "Standardkonstruktor" für main() in Space
26     }
27 }
```

Lösung: Unbekannt.