Programmiermethodik
LV-Nr.: 703017-_
Sommersemester 2013
Übungstest
3.6.2013
Dauer: 60 Minuten

Name: _____
Matrikelnummer: _____

Dieser Test enthält 14 Seiten (inklusive Deckblatt) und 4 Probleme. Bitte überprüfen Sie die Seitenzahl. Notieren Sie Name und Matrikelnummer auf dem Deckblatt.

Es sind *keine* Unterlagen erlaubt!

Sollte Ihnen der Platz für Antworten zu knapp werden, verwenden Sie bitte die Rückseite.

Bitte tragen Sie *nichts* in die Tabelle rechts ein!

Problem	Punkte	Erzielte Punkte
1	5	
2	10	
3	10	
4	15	
Total:	40	

1. (5 Punkte) Betrachten Sie das Programm aus Listing 1, welches 5 Fehler, die vom Compiler erkannt werden, enthält.

```
1  import java.io.*;
2
3  class Foo {
4
5    public static void main(String [] args) {
6      Foo p = new Foo();
7
8      boolean result = p.isOrdered(new int[]{1,2,4,5,9});
9      System.out.println(result);
10     result = isOrdered(new float[]{1f,2f,4.3f,5.2f,4.3f});
11     System.out.println(result);
12     int sum = p.mutiply(5,5);
13   }
14
15   static boolean isOrdered(int...xs) {
16     int temp = Integer.MIN_VALUE;
17     for(int x : xs) {
18         if( x => temp) { temp = x; }
19         else false;
20     }
21       return true;
22   }
23
24
25   int multiply(int x, int y) {
26     long result = x * y;
27     return result;
28   }
29
30   double multiply(double x, double y) {
31     double result = x * y;
32     return result;
33   }
34 }
```

Listing 1: Foo.java

Geben sie die Zeilennummer zu jedem Fehler, sowie eine Erläuterung desselbigen, wieder. Geben Sie weiters für jeden der erkannten 5 Fehler die korrekten Codefragmente wieder (jeweilige(r) Zeile/Block).

2. (10 Punkte) Kisten, Kisten, Kisten.

   Implementieren Sie ein Verwaltungssystem, das Gegenstände in Boxen verwalten kann. Eine Box kann dabei eine beliebige Anzahl an Gegenständen und kleineren Boxen beinhalten. Implementieren Sie zwei Klassen `Box` und `Item`, die das gegebene Interface `IBox` implementieren. Die Methode `getWeight` gibt das Gewicht einer Kiste bzw. eines Items zurück. Das Gewicht der Items wird explizit gespeichert, das Gewicht einer Kiste berechnet (wobei die Kiste selbst nichts wiegt). Die Methode `getMaxWeight` gibt das Gewicht des schwersten Items zurück. Implementieren Sie weiters die Methode `getItemsHeavierThan`, die alle Items in einer Kiste zurückliefert, die mehr wiegen als der Grenzwert der übergeben wird.

```
1  import java.util.Collection;
2
3
4  public interface IBox {
5
6    public double getWeight();
7
8    public double getMaxWeigth();
9
10   public Collection<Item> getItemsHeavierThan(double weight);
11
12 }
```

Listing 2: IBox.java

```
1  import java.util.Collection;
2
3
4  public class Box implements IBox {
5
6
7
8
9
10
11
12   public double getWeight() {
13
14
15
16
17
18
19
20
21   }
22   public double getMaxWeigth() {
23
24
25
26
27
28
29
30
```

```
32
   }
34
   public Collection<Item> getItemsHeavierThan(double weight) {
36
38
40
42
44
46
48
50
   }
52
54 }
```

Listing 3: Box.java

```
import java.util.Collection;
2
4 public class Item implements IBox {
6   private double weight;
8
10
12
14
   public Item(double weight) {
16   this.weight = weight;
   }
18
   public double getWeight() {
20
22
24
26
28 }
30 public double getMaxWeigth() {
32
```

```
34
36
38      }
40      public Collection<Item> getItemsHeavierThan(double weight) {
42
44
46
48
50
52
54
56      }
58  }
```

Listing 4: Item.java

3. (10 Punkte) Überladen, Überschreiben, ...

Betrachten Sie das Programm `Talk.java` - was gibt dieses Programm aus? Notieren Sie bitte jeweils die Zeilennummer und die entsprechende Ausgabe. Sie können ihre Antworten auch abkürzen (etwa Samantha: Carry wird zu S:C).

```
1  public class Talk {
       public static void main (String [] args) {
3          Miranda m1 = new Samantha();
           Miranda m2 = new Miranda();
5
           Carry c1 = new Miranda();
7          Carry c2 = new Samantha();

9          Samantha s1 = new Samantha();

11         Object o1 = new Samantha();
           Object o2 = new Carry();
13         Object o3 = new Miranda();

15         // Ausgabe

17         c2.print(new Miranda());
           ((Samantha) c2).print(new Miranda());
19         c1.print(new Samantha());
           m1.print(new Carry());
21         ((Miranda) c1).print(new Carry());
           s1.print((Miranda) m1);
23         ((Miranda) o3).print(new Samantha());
           s1.print(new Samantha());
25         m2.print(new Samantha());
           ((Samantha) o2).print(new Samantha());
27         ((Miranda) o1).print(new Miranda());
       }
29 }
```

Listing 5: Talk.java

```
class Carry {

    public void print(Carry p) {
        System.out.println("Carry: Carry\n");
    }

}
```

Listing 6: Carry.java

```
class Miranda extends Carry {

    public void print(Miranda l) {
        System.out.println("Miranda: Miranda\n");
    }

    public void print(Carry p) {
        System.out.println("Miranda: Carry\n");
    }
}
```

Listing 7: Miranda.java

```
class Samantha extends Miranda {
  public void print(Carry p) {
    System.out.println("Samantha: Carry\n");
  }

  public void print(Miranda l) {
    System.out.println("Samantha: Miranda\n");
  }

  public void print(Samantha b) {
    System.out.println("Samantha: Samantha\n");
  }
}
```

Listing 8: Samantha.java

4. Generics

(a) (5 Punkte) Betrachten Sie das Progamm aus Listing 9.

```
1  class Pair<X, Y> {
2
3    Y getValue() { return value; }
4
5    void setKey(X key) { this.key = key; }
6
7    X key;
8    Y value;
9  }
10
11 class Test {
12
13   void foo(Pair<String, Integer> p) {
14     int v = p.getValue();
15     p.setKey(String.valueOf(v));
16   }
17 }
```

Listing 9: Pair.java

Geben Sie die Klassen `Pair` und `Test` nach der Type-Erasure wieder.

(b) (5 Punkte) Betrachten Sie das Progamm aus Listing 10.

```
1  public class Wildcard {
2
3      //Note that we couldn't declare the 2D list as List<Object>
4
5       public static double foldLeft(List<?> list, int initialValue) {
6          Double sum = (double)initialValue;
7          for(Object e : list) {
8              switch(initialValue) {
9                  case 0: sum += (Double)e;
10                 case 1: sum *= (Double)e;
11             }
12
13         }
14         return sum;
15     }
16
17     public static void main(String [] args) {
18         List<Number> list = new LinkedList();
19         list.add(3);
20         list.add(null);
21         list.add(5.5);
22         System.out.println(foldLeft(list, 0));
23     }
24 }
```

Listing 10: Wildcard.java

- Warum der Kommentar? Warum kann `list` nicht als Liste von `Object` deklariert werden?

- Unter der Annahme, das Programm kompiliert, welches Problem könnte auftreten?

- Wie müsste der Typ von `list` abgeändert werden, damit keine Probleme auftreten können?

(c) (5 Punkte) Gegeben Sei das Programm aus Listing 11.

```
1  interface A { }
2
3  class B implements A { }
4
5  class C extends B { }
6
7
8  class Carpet<V extends A, B> {
9
10   public Carpet() {
11
12   }
13
14   public <X extends V> Carpet<?, ? extends V> method() {
15     // insert code here
16   }
17 }
```

Listing 11: Generics.java

Welche(s) der folgenden Statements kann/können in Zeile 15 eingefügt werden? Begründen Sie Ihre Antwort, insbesondere, warum ein Statement nicht erlaubt ist!

A `return new Carpet<V, X>();`

B `return new Carpet<X, V>();`

C `return new Carpet<A, B>();`

D `return new Carpet<B, C>();`

E `return new Carpet<A, C>();`