Programmiermethodik
LV-Nr.: 703017-2
Sommersemester 2013
Übungstest
3.6.2013
Dauer: 60 Minuten

Name: _____
Matrikelnummer: _____

Dieser Test enthält 13 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      int result = p.max(9,3);
8      System.out.println(result);
9      result = max(9.0,3);
10     System.out.println(result);
11     int sum = P.add(5,5);
12   }
13
14   static int max(int...xs) {
15     int temp;
16     for(int x : xs) {
17       if(x > temp) temp == x;
18     }
19     return temp;
20   }
21
22   int add(int x, int y) {
23     int result = x + y;
24     System.out.println(result);
25   }
26
27   double add(double x, double y) {
28     double result = x + y;
29     return result;
30   }
31 }
```

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 `Boxable` implementieren. Die Funktionen `getCount` und `getDepth` geben die Anzahl der Gegenstände (ohne Boxen!), bzw. die maximale Verschachtelungstiefe zurück. Ein Gegenstand in einer Box in einer Box hat dabei die Verschachtelungstiefe 2. Die Funktion `listItems` gibt eine Datenstruktur nach Wahl zurück, die alle Items enthält. Stellen Sie sicher, dass die von `listItems` zurückgegebene Datenstruktur keine Duplikate enthält. Sie können davon ausgehen, dass `equals` entsprechend implementiert ist.

```
1  import java.util.Collection;
2
3  public interface Boxable {
4
5    public int getCount();
6
7    public int getDepth();
8
9    public Collection<Item> listItems();
10
11 }
```

Listing 2: Boxable.java

```
1  import java.util.Collection;
2
3  public class Box implements Boxable {
4
5    public int getCount() {
6
7
8
9
10
11
12
13
14   }
15
16   public int getDepth() {
17
18
19
20
21
22
23
24
25   }
26
27   public Collection<Item> listItems() {
28
29
30
```

```
32
34
36
    }
38
}
```

Listing 3: Box.java

```
1  import java.util.Collection;
3
   public class Item implements Boxable {
5
     public int getCount() {
7
9
11
13
15   }
17   public int getDepth() {
19
21
23
25
     }
27   public Collection<Item> listItems() {
29
31
33
35
37   }
39 }
```

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();
           Carry c3 = new Carry();
9
           Samantha s1 = new Samantha();
11
           Object o1 = new Samantha();
13         Object o2 = new Carry();
           Object o3 = new Miranda();
15
           // Ausgabe
17         c3.print(new Samantha());
           ((Samantha) c2).print(new Samantha());
19         c1.print(new Samantha());
           s1.print(new Miranda());
21         new Samantha().print(new Miranda());
           ((Carry) o2).print(new Miranda());
23         ((Samantha) m1).print(new Miranda());
           s1.print((Carry) c2);
25         m2.print(new Carry());
           ((Samantha) o3).print(new Samantha());
27         ((Carry) o1).print(new Carry());
       }
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<K, V> {
2
3    K getKey() { return key; }
4
5    void setKey(K key) { this.key = key; }
6
7    K key;
8    V value;
9  }
10
11 class Test {
12
13   void foo(Pair<String, Integer> p) {
14     String s = p.getKey();
15     p.setKey("foo");
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.

```
class Wildcard {

   //Note that we couldn't declare list as List<List<Object>>

    public static List flatten2DList(List<List<?>> list) {
        List flatList = new LinkedList();
        for(List sublist : list) {
            for( Object e : sublist) {
                flatList.add(e);
            }
        }
        return flatList;
    }

    public static void main(String [] args) {
        List<List<?>> list1 = new LinkedList<List<?>>();
        LinkedList<Integer> list2 = new LinkedList();
        list2.add(3);
        list2.add(null);
        LinkedList<Double> list3 = new LinkedList();
        list3.add(5.5);
        list1.add(list2);
        list1.add(list3);
        System.out.println(flatten2DList(list1));
    }
}
```

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?

- Unter der Annahme, dass der Parameter `list` in der Methode `flatten2DList` den Typ `List<List<? extends Number>>` bekommt, was müsste in der `main` Method angepasst werden, damit das Programm kompiliert?

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

```
1  class A { }
2
3  class B extends A { }
4
5  class C extends B { }
6
7
8  public class Carpet<V extends 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<X>();`

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

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

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

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