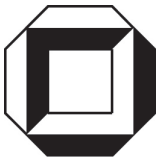

Universität Karlsruhe (TH)

Lehrstuhl für Programmierparadigmen

Fortgeschr. Objektorientierung SS 2009		http://pp.info.uni-karlsruhe.de/
Dozent:	Prof. Dr.-Ing. G. Snelting	snelting@ipd.info.uni-karlsruhe.de
Übungsleiter:	Andreas Lochbihler	lochbihl@ipd.info.uni-karlsruhe.de
	Dennis Giffhorn	giffhorn@ipd.info.uni-karlsruhe.de

Blatt 2 | Ausgabe: 30.04.2009 | Besprechung: 06.05.2009

1. **Zerbrechliche Subklassen**

   Folgende Klasse implementiert eine Menge, die zählt, wie viele Elemente eingefügt wurden. Welche Probleme entstehen durch die Verwendung von Vererbung? Wie könnte man diese beheben?

```
1  class CountingHashSet extends HashSet {
2
3      private int addCount = 0;
4
5      public CountingHashSet() {
6      }
7
8      public CountingHashSet(Collection c) {
9          super(c);
10         addCount += c.size();
11     }
12
13     public int getAdditionCount() {
14         return addCount;
15     }
16
17     public boolean add(Object e) {
18         addCount++;
19         return super.add(e);
20     }
21
22     public boolean addAll(Collection c) {
23         addCount += c.size();
24         return super.addAll(c);
25     }
26 }
```

   Hinweis: Die Klasse `HashSet` aus der Java-Bibliothek könnte wie folgt implementiert sein:

```
1  public class HashSet extends AbstractSet {
2      public HashSet() {
3          this(16);
4      }
5
6      public HashSet(Collection c) {
7          this(Math.max((int) (c.size()/.75f + 1), 16));
8          addAll(c);
9      }
10
11     ...
12 }
13
14 public abstract class AbstractSet
15     extends AbstractCollection implements Set { ... }
16
17 public interface Set {
18     public boolean add(Object o);
19     public boolean addAll(Collection c);
20     ...
21 }
```

2. **Variable Methodenparameteranzahl**

   Seit Version 1.5 erlaubt es Java, eine unspezifizierte Anzahl an Methodenparametern zu übergeben. Dies wird dann durch '...' angezeigt. Betrachten Sie unten angehängtes Codestück. Gibt der Compiler Meldungen aus? Was ist das Resultat zur Laufzeit?

```
1  class A {
2      public void f(String msg, int... is) {
3          System.out.println("in A.f: "+msg);
4      }
5
6      public void g(String msg, int[] is) {
7          System.out.println("in A.g: "+msg);
8      }
9  }
10
11 class B extends A {
12     public void f(String msg, int[] is) {
13         System.out.println("in B.f: "+msg);
14     }
15
16     public void g(String msg, int... is) {
17         System.out.println("in B.g: "+msg);
18     }
19
20 }
21
22 public class Parameters {
23
24     public static void main(String args[]) {
25         int[] numbers = {1,3,5};
26         B b = new B();
27         A a = new B();
28         b.f("Array",numbers);
29         b.f("Parameters",2,4,6);
30         a.f("Array",numbers);
31         a.f("Parameters",2,4,6);
32
33         b.g("Array",numbers);
34         b.g("Parameters",2,4,6);
35         a.g("Array",numbers);
36         a.g("Parameters",2,4,6);
37
38     }
39
40 }
```

3. **Subobjekte**

Betrachten Sie die Klassen-Hierarchie in folgendem C++ Programm:

```
1  class A {
2      public: int i;
3  };
4
5  class B : public A { };
6
7  class C : public A { };
8
9  class D1 : public A, public B, public C { };
10
11 class D2 : public C, public B, public A { };
```

(a) Zeichnen Sie den Klassengraphen, den Subobjektgraphen und das Objektlayout von $D1$ und $D2$.

(b) Wieviele A Subobjekte sind in einem $D1$ bzw. $D2$ Objekt enthalten?

(c) Durch Typcasts können Sie Mehrdeutigkeiten beim Zugriff auf Members beseitigen. Schreiben Sie die Casts auf, die notwendig sind, um auf das Member i aller A Subobjekte eines $D1$ bzw. $D2$ Objekts zuzugreifen.