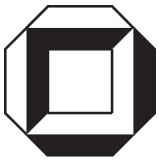

Universität Karlsruhe (TH)

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 5	Ausgabe: 20.05.2009	Besprechung: 27.05.2009

1. **vtables / statische Information und final overrider**

   Betrachten Sie folgendes Programm:

```
1  class A {
2  public: virtual void foo() {}
3  };
4  class B: public A {
5  public: virtual void foo() {}
6  };
7  class C: public A {
8  };
9  class D: public C, public B  {
10 };
11
12 int main() {
13   C* c = new D();
14   c->foo();
15 }
```

   (a) Erstellen Sie die vtables für die Subobjekte in D.

   (b) Welche $foo()$-Methode ruft `main()` auf?

(c) Betrachten Sie nun folgendes, leicht modifiziertes Programm. Wie ändert sich das Verhalten?

```
1  class A {
2  public: virtual void foo() {}
3  };
4  class B: public virtual A {
5  public: virtual void foo() {}
6  };
7  class C: public virtual A {
8  };
9  class D: public C, public B  {
10 };
11
12 int main() {
13   C* c = new D();
14   c->foo();
15 }
```

(d) Fügen Sie nun in das Programm in (c) eine Methodendefinition von foo in C ein. Was meldet der Compiler? Ändert sich das Resultat, wenn man die `main`-Methode auskommentiert? Erläutern Sie dies anhand der vtables.

2. **Auflösen überladener Methoden (Java)**

(a) Kompiliert das folgende Programm? Wenn ja, welche Methode wird bei Ausführung aufgerufen?

```
1  class Test {
2          public static void main(String[] args) {
3                  new A().f(null);
4          }
5  }
6  class A {
7          void f(Object o) {}
8          void f(A a) {}
9  }
```

(b) Kompiliert das folgende Programm? Wenn ja, welche Methode wird bei Ausführung aufgerufen? Was passiert, wenn B Unterklasse von A ist?

```
1  class Test {
2          public static void main(String[] args) {
3                  new A().f(null);
4          }
5  }
6  class A {
7          void f(A a) {}
8          void f(B b) {}
9  }
10 class B {}
```

(c) Erklären Sie das Verhalten des Compilers für beide Programme.

3. **Wechselwirkung von Überladung und Vererbung (Java)**

   (a) Welche Methoden werden bei Ausführung des folgenden Programms aufgerufen?

```
 1  class A {}
 2
 3  class B extends A {}
 4
 5  class C {
 6     void f(B b) { }
 7  }
 8
 9  class D extends C {
10     void f(A a) { }
11  }
12
13  class Test {
14     public static void main(String[] args) {
15        A a=new A();
16        B b=new B();
17        D d=new D();
18        C c=d;
19        d.f(a);
20        c.f(b);
21        d.f(b);
22     }
23  }
```

   (b) Der Compiler aus JDK 1.3 meldet einen Syntaxfehler in Zeile 21. Dieser "Fehler" verschwand durch eine Änderung der Sprachdefinition. Geben Sie mögliche Ursachen für diese Änderung an.

4. **(Optional: Integer-Autoboxing)**

   Seit Java 1.5 gibt es sogennantes *Integer-Autoboxing* bzw. *-Autounboxing*. Dabei wird ein `int`, wenn er als Objekt benötigt wird (z.B. beim Einfügen in einen Container) automatisch in ein `Integer`-Objekt umgewandelt. Andererseits wird ein `Integer`-Objekt, wenn es als primitiver Wert benötigt wird (z.B. bei mathematischen Operationen), automatisch in einen `int` Wert umgewandelt.

Betrachten Sie nun folgenden Java-Code:

```
1  public class IntegerEquals {
2
3      public static void foo(Integer i1, Integer i2) {
4          System.out.println("in foo(Integer,Integer)");
5      }
6
7      public static void foo(Integer i1, int i2) {
8          System.out.println("in foo(Integer,int)");
9      }
10
11     public static void foo(int i1, Integer i2) {
12         System.out.println("in foo(int,Integer)");
13     }
14
15     public static void foo(int i1, int i2) {
16         System.out.println("in foo(int,int)");
17     }
18
19     public static boolean equals1(Integer i1, Integer i2) {
20         return (i1 == i2);
21     }
22
23     public static boolean equals2(Integer i1, int i2) {
24         return (i1 == i2);
25     }
26
27     public static boolean equals3(int i1, Integer i2) {
28         return (i1 == i2);
29     }
30
31     public static boolean equals4(int i1, int i2) {
32         return (i1 == i2);
33     }
34
35
36     public static void main(String[] args) {
37         Integer i1 = new Integer(42);
38         Integer i2 = new Integer(42);
39
40         foo(i1,i2);
41
42         System.out.println(equals1(i1,i2));
43         System.out.println(equals2(i1,i2));
44         System.out.println(equals3(i1,i2));
45         System.out.println(equals4(i1,i2));
46
47     }
48 }
```

(a) Was passiert beim Aufruf von $foo(i1, i2)$? Was passiert, wenn man die erste Methodendefinition von foo auskommentiert?

(b) Welche Resultate liefern die verschiedenen `equals`-Methoden?

(c) Jedoch gibt es noch ein paar mehr Feinheiten, die man beachten muss, z.B. bei der Arbeit mit Containern. Betrachten Sie dazu folgendes Codefragment:

```
1   int a = 12, b = 12;
2   Integer c = new Integer(12),d = new Integer(12);
3   Integer e = a, f = b;
4
5   a == b;
6   c == d;
7   a == c;
8   e == f;
9   a == f;
10
11  LinkedList<Integer> list = new LinkedList<Integer>();
12  list.add(0,a);
13  list.add(1,b);
14  list.add(2,c);
15  list.add(3,d);
16  list.add(4,e);
17  list.add(5,f);
18
19  list.get(0) == list.get(1);
20  list.get(2) == list.get(3);
21  list.get(0) == list.get(2);
22  list.get(4) == list.get(5);
23  list.get(0) == list.get(5);
```

i. Welche Ergebnisse liefern die Gleichheitstests? Wie erklären Sie sich die Resultate? Ändern sich die Resultate, wenn 12 durch 12000 ersetzt wird?

ii. Welchen Typ hat die Variable *result* im untenstehenden Codestück? Wie sieht die Liste danach aus? Wie sieht es aus, wenn man in remove statt `new Integer(0)` einfach 0 schreibt?

```
1   LinkedList<Object> objectList =
2     new LinkedList<Object>();
3   objectList.add(1);
4   objectList.add(0);
5   ? result = objectList.remove(new Integer(0));
```