Universität Karlsruhe (TH)

Lehrstuhl für Programmierparadigmen

Fortgeschr. Objektorientierung SS 2008 http://pp.info.uni-karlsruhe.de/
Dozent: Prof. Dr.-Ing. G. Snelting snelting@ipd.info.uni-karlsruhe.de
Übungsleiter: Daniel Wasserrab wasserra@ipd.info.uni-karlsruhe.de
Andreas Lochbihler lochbihl@ipd.info.uni-karlsruhe.de

Übungsblatt 3 Ausgabe: 19.5.2008 Besprechung: 21.5.2008

1. **vtables / Type Casts**

   Betrachten Sie folgende Klassenhierarchie:

```
1   class A {
2     virtual void f() { ... }
3     virtual void g() { ... }
4   };
5   class B : public A {
6     virtual void f() { ... }
7   };
8   class C {
9     virtual void g() { ... }
10    virtual void h() { ... }
11  };
12  class D : public B, public C {
13    virtual void f() { ... }
14  };
15  class E : public virtual D, public virtual A {
16  };
```

   (a) Erstellen Sie die vtables mit Methodennamen und Subobjekt-Deltas

   (b) Wie verändern sich die v-tables, wenn E D nicht virtuell, sondern nichtvirtuell erbt?

   (c) Schreiben Sie den Code für die folgenden Funktionsaufrufe auf (Notation wie im Skript, Seite 51ff):

```
1   D *d=new D();
2   d->f();
3   d->g();
4   d->h();
```

(d) Für welchen der folgenden Type-Casts muss Code erzeugt werden? Schreiben Sie ihn in ähnlicher Notation wie zuvor auf.

```
1          D* d=new D();
2          E* e=new E();
3          A* pa; B* pb; C* pc; D* pd; E* pe;
4          pa=d; pd=(D*)pa;
5          pb=d; pd=(D*)pb;
6          pc=d; pd=(D*)pc;
7          pa=e; pe=(E*)pa;
8          pb=e; pe=(E*)pb;
9          pc=e; pe=(E*)pc;
```

(e) Das Subobjekt-Layout von E lässt die Vermutung zu, man könnte auf die virtuellen D- und A-Subobjekte nicht nur über den in E enthaltenen Pointer auf diese Subobjekte zugreifen, sondern auch über ein entsprechendes δ. Warum geht das nicht? Was passiert wenn ein Objekt einer Klasse

```
1          class F : public virtual A,
2                    public virtual D,
3                    public virtual E { };
```

erst nach E und dann nach D gecastet wird?

2. **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.

3. **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
7  class A {
8          void f(Object o) {}
9          void f(A a) {}
10 }
```

(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
7  class A {
8          void f(A a) {}
9          void f(B b) {}
10 }
11
12 class B {}
```

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

4. **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.

5. **Überladung des Rückgabewertes**

Bei der Überladungsauflösung wird in ADA der Typ des Rückgabewertes mit berücksichtigt, in Java und C++ wird er ignoriert.

(a) Finden Sie Beispiele, in denen das Verhalten von ADA sinnvoll angewendet werden kann.

(b) Diskutieren Sie Vor- und Nachteile beider Möglichkeiten.