Übungen zu Objektorientierte Programmierung WS 98/99

1) Können in einem Modul eine Methode und eine Prozedur mit den gleichen Namen deklariert werden?

   Ja, da eine Methode lokal zur Klasse ist, zu der sie gehört. Es gibt deshalb keinen Namenskonflikt mit global deklarierten Namen.
2) Kann eine Methode an eine Klasse gebunden werden, die in einem anderen Mopul deklariert ist?
   Nein. Die Lokalität des Programmcodes ist ein wichtiges Prinzip, das die Wartung von Software erleichtert. Wenn Methoden einer Klasse über verschiedene Module verteilt wären, würde das dem Prinzip der Lokalität widersprechen.
3) Kann eine Meldung an ein Zeigerobjekt gesendet werden, wenn der formale Empfänger einer Methode vom Recordtyp ist?

```
TYPE
   Ptr = POINTER TO Rec;
   Rec = RECORD ... END;
VAR
   p: Ptr;
   r: Rec;
PROCEDURE (VAR r: Rec) M; ... END M;
PROCEDURE (p: Ptr) M1; ... END M1;
... p.M ...    <= legal
... r.M1 ...    <= nicht legal
```

   Ja. Das Record, auf das p zeigt, wird als Referenzparameter an die Methode M übergeben. Andererseits kann man eine Meldung nicht an ein Rekordobjekt senden, wenn der formale Parameter ein Zeiger ist. Ein Record kann nicht an einen Zeiger übergeben werden. Wenn sowohl Variablen eines Zeiger- und eines Recordtyps die Empfänger von Meldungen sein sollen, muß man als formalen Empfänger einen Referenzparameterrecord haben.
4) Warum kann man geerbte Instanzvariablen in einer Unterklasse nicht überschreiben?
   Nein. Angenommen die Basisklasse A hat eine Instanzvariable a vom Typ INTEGER und die Unterklasse B hat eine Instanzvariable a vom Typ CHAR. Dann könnte der Compiler keine Typprüfung für eine Anweisung der folgenden Art durchführen: a.f := a.f + 1. Wenn a vom dynamischen Typ A ist, ist a.f vom Typ INTEGER und die Addition legal, wenn a vom dynamischen Typ B ist, ist a.f vom Typ CHAR und die Addition nicht legal. Typprüfung müßte deshalb zur Laufzeit durchgeführt werden.
5) Kann eine Klasse von mehreren Basisklassen abgeleitet werden?
   Nein, nicht in Oberon-2.
6) Kann eine Unterklasse auf Instanzvariablen und Methoden ihrer Oberklassen zugreifen, wenn diese in einem anderen Modul deklariert aber nicht exportiert sind?
   Nein. In Oberon-2 passiert "Information hiding" an den Modulgrenzen. Auch Unterklassen können diese Grenzen nicht überschreiten.
7) Muß eine exportierte Methode jedesmal, wenn sie überschrieben wird, wieder exportiert werden?
   Ja.
8) Ist die Reihenfolge der Typtest in der WITH-Anweisung signifikant?
   Beispiel:

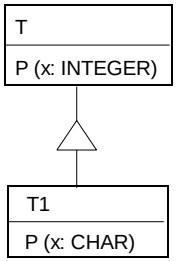
```
   T = RECORD END; T1 = RECORD (T) ... END; T2 = RECORD (T1) ... END;
   P = POINTER TO T; P1 = POINTER TO T1; P2 = POINTER TO T2;

   VAR p0: P; p1: P1; p2: P2;

   NEW(p1); NEW(p2);
   p0 := p2;
   WITH p0: P1 DO Out.String("p0 IS P1")
   | p0: P2 DO Out.String("p0 IS P2")
   ELSE Out.String("p0 is neither P1 nor P2")
   END
```

   Ja, Ausgabe in diesem Beispiel "p0 IS P1".
9) Wo findet man Information, was die Nummer eines Laufzeitfehlers/Traps bedeutet?

   In der Datei OberonErrors.Text stehen die Erklärungen der Fehlermeldungen des Compilers, Erklärungen zu den Limitierungen und der Implementierung des Compilers, Erklärungen der Warnungen, Erklärungen zu internen Fehlern und Erklärungen der Laufzeitfehlernummern.
10) Wenn man nur den Zeiger auf einen Typ exportiert, nicht aber den Typ selber, kann man dann von importierenden Modulen nur auf die typgebundenen Prozeduren, die an einen Empfänger mit Zeigertyp gebunden sind, zugreifen?
    Beispiel:

```
MODULE Test;
IMPORT Out;

TYPE
   T* = POINTER TO TDesc;
   TDesc = RECORD END;

PROCEDURE (VAR t: TDesc) A*; BEGIN Out.String("A") END A;
PROCEDURE (t: T) B*; BEGIN Out.String("B") END B;

MODULE Test2;
IMPORT Test;

PROCEDURE Do*;
   VAR t: Test.T;
BEGIN
   NEW(t);
   t.A;    <= erlaubt?
   t.B;
END Do;
END Test.
```

    Nein, man kann auf alle typgebundenen Prozeduren zugreifen, die exportiert wurden, egal ob der Typ des formalen Receivers vom Recordtyp oder Zeigertyp ist.
11) Sind exportierte Felder eines nichtexportierten Typs in einem importierenden Modul sichtbar?
    Nein, ein Feld ist nur sichtbar, wenn auch der enthaltende Typ sichtbar (d.h. exportiert) ist.
12) Wenn man nur den Zeiger auf einen Typ exportiert, nicht aber den Typ selber, kann man dann in importierenden Modulen eine Erweiterung dieses Typs definieren?
    Nein, der Basistyp wurde nicht exportiert und ist somit nicht sichtbar.
13) Sind Methoden, die in einer Klasse mit "-" (read only) markiert wurden, bei der Redefinition in einer Unterklasse überschreibbar?
    Ja, die Exportmarkierung mit "-" bietet keine Einschränkung in Bezug auf Überschreibbarkeit (wäre eingeschränkte Vererbung).
14) Kann man, wenn eine Klasse A eine nichtexportierte Methode P besitzt, in einem anderen Modul in einer von A abgeleiteten Klasse B eine Methode P mit beliebiger Parameterliste definieren? Ändert sich etwas an ihrer Antwort, wenn A und B im selben Modul implementiert werden?
    Ja, man sieht die Methode P der Basisklasse ja nicht (auch deren Parameterliste nicht) und muß somit bei seiner Wahl der Prozedurnamen und -parameterlisten nicht Rücksicht nehmen.
    Ja, dann ist die Methode P der Klasse A in der Klasse B ja sichtbar. Beim Überschreiben muss man deswegen die gleiche Parameterliste angeben.
15) Kann der dynamische Typ eines Empfängerparameters von seinem statischen Typ abweichen? Geben Sie mögliche Codebeispiele an und diskutieren Sie die Möglichkeiten.
    Ja, aufgrund der Zuweisungskompatibilität bei Zeigern kann an eine Zeigervariable vom Typ P auch eine Zeigervariable vom Typ P1 zugewiesen werden, wenn P = POINTER TO T, P1 = POINTER TO T1 und T1 eine Erweiterung von T ist. Methoden werden an den dynamischen Typ des Empfängers gebunden. Damit kann man polymorphes Verhalten implementieren.
    Bei einem Super-Call ruft man eine statisch gebundene Methode auf (Methode, die zum Basistyp des statischen Typs des Empfängers gehört). Insoferne unterscheiden sich in diesem Fall auch der statische Typ des Empfängers und die Klasse, zu der die Methode gehört.
16) Warum darf man beim Überschreiben einer Methode die Parameterliste nicht ändern?




```
VAR p: T;
...
p.P(3)    <= Problem, wenn dynamischer Typ von p T1 ist.
```

Wenn der dynamische Typ von p nicht mehr nur T sein kann, sondern auch eine Erweiterung von T (z.B. T1), und in einer Erweiterung die Parameterliste von P geändert ist, müßte der Compiler beim Prüfen der Parameterliste den dynamischen Typen des Empfängers des Methodenaufrufs betrachten.