

# Concepts of Programming Languages (INFOTECH)

Summer Semester 2007  
Prof. Plödereder, Stefan Staiger

## Assignment #3

Please submit your solution on Friday, July 20th 2007 in class or send it via email to [staiger@informatik.uni-stuttgart.de](mailto:staiger@informatik.uni-stuttgart.de). (If you want to submit earlier, please do so.) Please submit either plain text or PDF, but not MS Word files.

### 1 Namebinding and Method Calls

The following source code is written in an object-oriented language.

```
class T is
  ...
  method M (X : T) is begin Print ("T"); end M;
end class T;

class NT inherits T is
  ...
  method M (X : T) is begin Print ("NT"); end M; -- redefinition!
  method M2 (X : NT) is begin ... end M2;
end NT;

OT : T;
ONT : NT := new NT;
OT := new NT;
...
OT.M (OT);    -- (1)
OT.M2 (ONT);  -- (2)
ONT.M (OT);   -- (3)
ONT.M2 (ONT); -- (4)
```

**a.** Assume that the language uses dynamic name binding for method calls (like Smalltalk).

Are the calls at (1) to (4) legal calls? If so, which output will be produced by each call (assume that the calls are alternative implementations, so they do not influence each other)? If not, what kind of error diagnostic do you expect?

**b.** In the following, the language uses static name binding for method calls (like C++, Java, Eiffel).

Repeat part a. Which results are the same? Which differ? Please state the reason for the observed behaviour.

**c.** What are the advantages of static name binding for method calls?

General questions:

**d.** Explain the difference between overloading and redefinition.

- e. For which parts (return type, type of the controlling object, type of other parameters) of a signature does C++ use covariant, contravariant or invariant inheritance? Take the following method declaration as an example:

```
class C {
    return_type methodname (param_type1 param1, param_type2 param2);
};
```

## 2 Exceptions

Examine this program written in Ada 95:

```
01 with Ada.Text_IO; use Ada.Text_IO;
02 procedure Main is
03     E1, E2, E3 : exception;
04     procedure Q is
05     begin
06         raise E2;
07     exception
08         when E1 => Put_Line ("Q: E1"); raise E3;
09         when others => Put_Line ("Q: other exception"); raise E1;
10     end Q;
11     procedure P is
12     begin
13         begin
14             Q;
15         exception
16             when E1 => Put_Line ("P/1: E1"); Q; raise E2;
17         end;
18         begin
19             Q;
20         exception
21             when others => Put_Line ("P/2: other exception"); raise E1;
22         end;
23     exception
24         when E1 => Put_Line ("P/3: E1"); raise;
25                 Put_Line ("P/3: after raise");
26         when E2 => Put_Line ("P/3: E2"); raise E3;
27     end P;
28 begin
29     P;
30 exception
31     when E1 => Put_Line ("Main: E1");
32     when E2 => Put_Line ("Main: E2");
33     when E3 => Put_Line ("Main: E3");
34     when others => Put_Line ("Main: other exception");
35 end Main;
```

Note: If an exception is raised (or re-raised) inside an exception handler, the same list of handlers is not checked again. Instead the exception is propagated to the outside of the current block. It may then be caught by a handler of an outer block.

In what order are the statements in this program executed? What is the output of the program?