Compiler Practical 2013

Inheritance, Static and Dynamic Binding

Berthold Hoffmann (B. Gersdorf, T. Röfer)
hof@informatik.uni-bremen.de
Cartesium 2.48

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

Universität Bremen

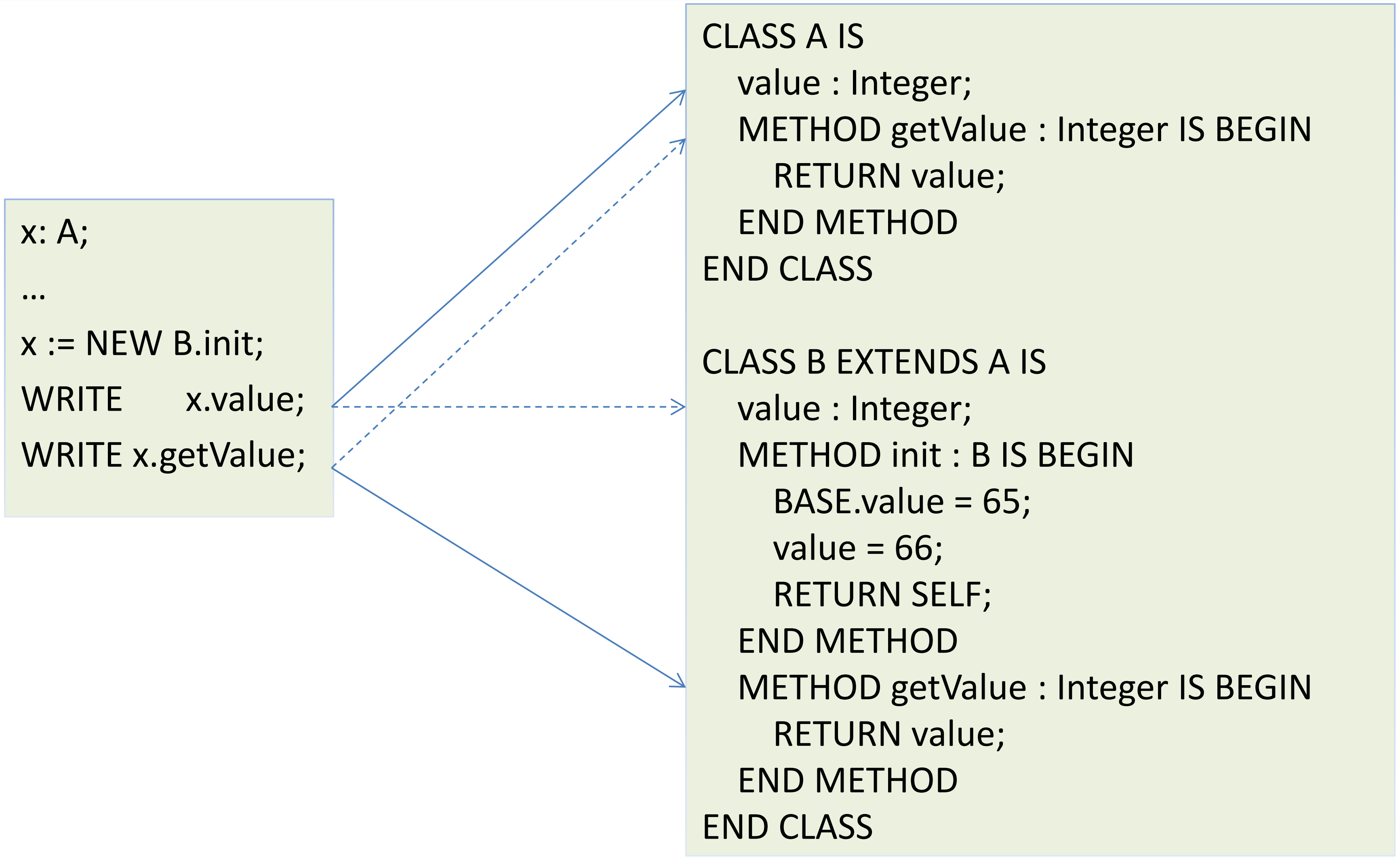
Structure

1. Late Binding (Dynamic Dispatch)
2. Virtual Method Tables
3. Context Analysis
4. Synthesis
5. *BASE*
6. Bonus Task: Type Checking and Type Casts at Runtime

Late Binding (Dynamic Dispatch)

- Virtual methods
  - The actual type of the receiver object, not ist base type, determines which method is called
  - From now on, *all* methods in LOOP are virtual
- Late Binding
  - It is determined only at runtime which (virtual) method is called
- Virtual Method Tables
  - Objects contain hidden references to the virtual method table of their class.
  - In this table, addresses of the methods of the class are stored.

Early or Late Binding?

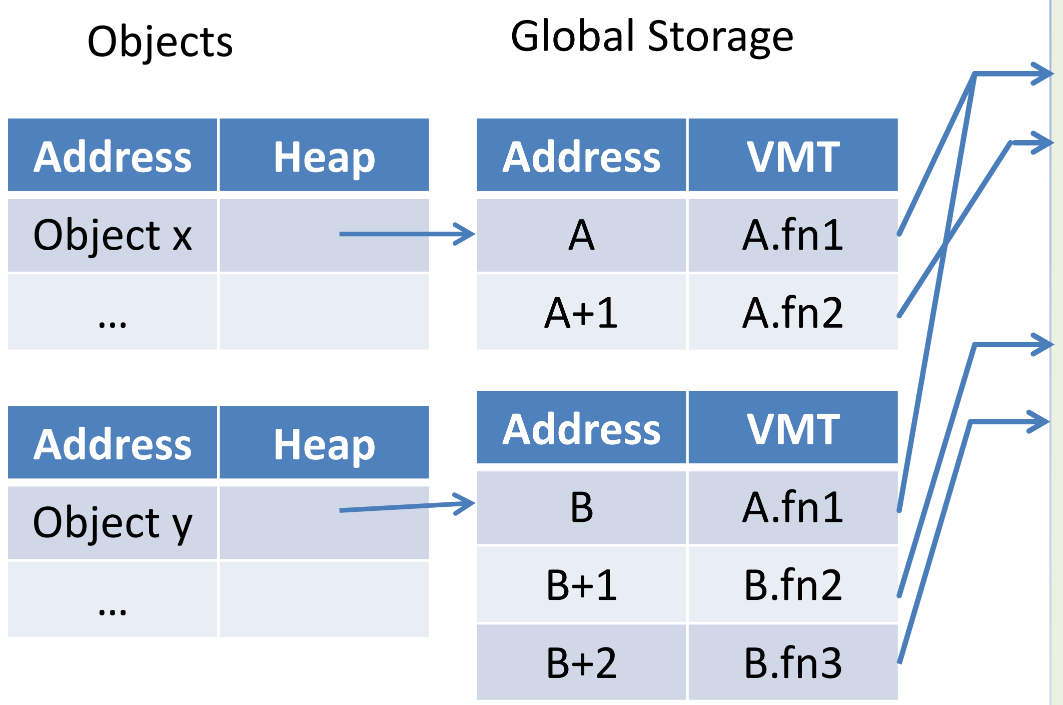


```
x: A;
…
x := NEW B.init;
WRITE       x.value;
WRITE x.getValue;
```

```
CLASS A IS
    value : Integer;
    METHOD getValue : Integer IS BEGIN
        RETURN value;
    END METHOD
END CLASS

CLASS B EXTENDS A IS
    value : Integer;
    METHOD init : B IS BEGIN
        BASE.value = 65;
        value = 66;
        RETURN SELF;
    END METHOD
    METHOD getValue : Integer IS BEGIN
        RETURN value;
    END METHOD
END CLASS
```

Virtual Method Tables (VMT)



Objects

Address	Heap
Object x	→ A
…	

Address	Heap
Object y	→ B
…	

Global Storage

Address	VMT
A	A.fn1
A+1	A.fn2

Address	VMT
B	A.fn1
B+1	B.fn2
B+2	B.fn3

```
CLASS A IS
   METHOD fn1 …
   METHOD fn2 …
END CLASS
CLASS B EXTENDS A IS
   METHOD fn2 …
   METHOD fn3 …
END CLASS
…
x,y: A;
x := NEW A;
x.fn1;
y := NEW B;
y.fn1;
y.fn2;
y.fn3; | compilation error
```

Context Analysis: Overriding

- If a method overrides another one, their signatures must be identical
  - Otherwise, this would be *overloading*, which is not supported in LOOP
- What happens if a method overrides a variable, or vice versa?
  - Either, this is *forbidden*, as it is overloading,
  - Or, it depends whether in the class of the accessing reference, the method or attribute is *visible*.

Context Analysis: Method Numbers

- Every method is associated with a number
  - The index in ist VMT
  - A new attribute in class *MethodDeclaration*
- For new methods, numbering starts after the last method number of the base class
  - In class *Object*, numbering starts at 0
- In case of overriding, the existing method number is reused

Synthesis: Generating VMTs

- For every class, a VMT must be generated
- Preparation
  - Generate Java array of *MethodDeclarations* of the actual class
  - Have it filled by the base classes and the actual class
  - Every entry contains the latest overriden method in the most derived class

```
CLASS A IS
   METHOD fn1 …
   METHOD fn2 …
END CLASS
CLASS B EXTENDS A IS
   METHOD fn2 …
   METHOD fn3 …
END CLASS
```

Offset	VMT
0	A.fn1
1	~~A.fn2~~ B.fn2
2	B.fn3

Synthesis: Code, Object instances

- Code generation
  - Theaddress of the table is labelled with *<class>:*
  - Then generate *DAT 1, <class>_<method>* for every method
- Object instances
  - *NEW* enters the address of the VMT at the address of the object (relative address 0)
  - Attributes start at relative adddress 1
  - *ClassDeclaration.HEADER_SIZE* = 1;

Synthesis: Method Call

- The address of the object is needed twice:
  - As parameter *SELF*
  - For determining the address of the VMT
- Not every method call is bound lately …

BASE

- Access to attributes and methods of the base class in a method body
- *BASE* and *SELF*
  - *BASE* ist the same local variable as *SELF*, i.e., both lie at the same stack address
  - The type of *SELF* is the actuall class, the type of *BASE* is the base class
  - *BASE* must be an R-value
- Method calls via *BASE* are not bound lately!

```
CLASS A IS
    METHOD a IS
    BEGIN
        WRITE 65;
    END METHOD
END CLASS

CLASS B EXTENDS A IS
    METHOD a IS
    BEGIN
        BASE.a;
        WRITE 66;
    END METHOD
END CLASS
```

Bonus: Type Checking and …

- <expr> ISA <class>
  - Is <expr> of the type of <class>? (or of one of its subclasses?)
- <class>(<expr>)
  - Yields NULL if <expr> is not of type <class>, and is the identity otherwise

```
CLASS Main IS
  METHOD main IS
    a : Object;
    b : Main;
  BEGIN
    a := SELF;
    IF a ISA Main THEN
      b := Main(a);
    END IF
  END METHOD
END CLASS
```

Bonus: …Type Casts at Runtime

- Type is the address of the VMT
- Every VMT has a pointer to the VMT of ist base class
  - Address of the base class of *Object* is 0
- *ISA* follows these pointers
- It holds true that *NULL ISA Object = TRUE*

5%

```
CLASS Main IS
  METHOD main IS
    a : Object;
    b : Main;
  BEGIN
    a := SELF;
    IF a ISA Main THEN
      b := Main(a);
    END IF
  END METHOD
END CLASS
```