Method Lookup

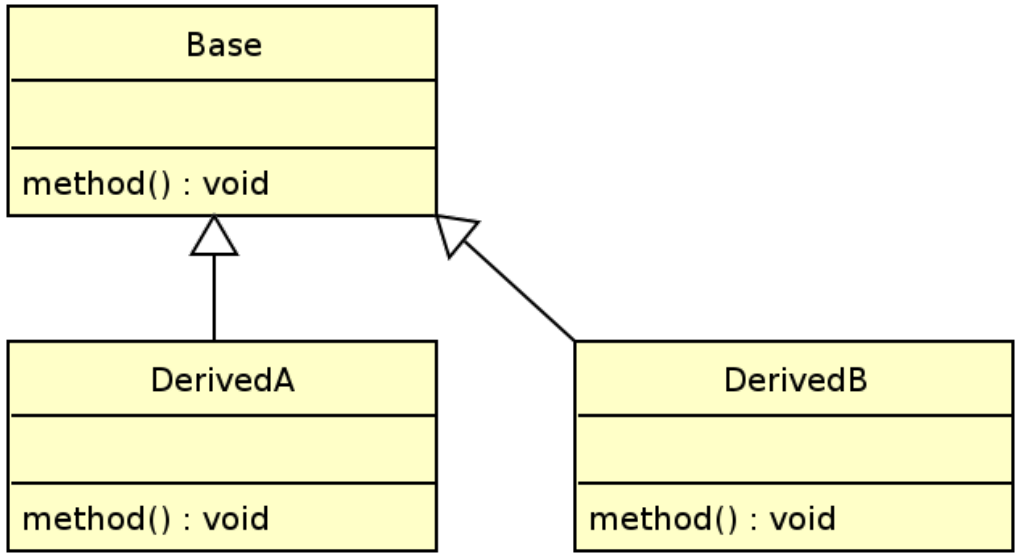
Jan Vraný

Department of Computer Science and Engineering
Czech Technical University In Prague
Faculty of Electrical Engineering

October 9, 2008

Outline

Case Study I: Method Lookup




```
Base x←new DerivedA()
x.method()
```

Case Study I: Java

```
1  public class Lookup {
2      public static class Base {
3          public void method() {
4              System.out.println("Base.method");
5          }
6      }
7      public static class DerivedA extends Base {
8          public void method() {
9              System.out.println("DerivedA.method");
10         }
11     }
```

Case Study I: Java

```
1  public class Lookup {
2      public static class Base {
3          public void method() {
4              System.out.println("Base.method");
5          }
6      }
7      public static class DerivedA extends Base {
8          public void method() {
9              System.out.println("DerivedA.method");
10         }
11     }
12     public static void main ( String[] argv ) {
13         Base x = new DerivedA();
14         DerivedA y = new DerivedA();
```

Case Study I: Java

```
1  public class Lookup {
2      public static class Base {
3          public void method() {
4              System.out.println("Base.method");
5          }
6      }
7      public static class DerivedA extends Base {
8          public void method() {
9              System.out.println("DerivedA.method");
10         }
11     }
12     public static void main ( String[] argv ) {
13         Base x = new DerivedA();
14         DerivedA y = new DerivedA();
15         x.method();
```

Case Study I: Java

```
1  public class Lookup {
2      public static class Base {
3          public void method() {
4              System.out.println("Base.method");
5          }
6      }
7      public static class DerivedA extends Base {
8          public void method() {
9              System.out.println("DerivedA.method");
10         }
11     }
12     public static void main ( String[] argv ) {
13         Base x = new DerivedA();
14         DerivedA y = new DerivedA();
15         x.method();
16         y.method();
```

Case Study I: Java

```
1  public class Lookup {
2      public static class Base {
3          public void method() {
4              System.out.println("Base.method");
5          }
6      }
7      public static class DerivedA extends Base {
8          public void method() {
9              System.out.println("DerivedA.method");
10         }
11     }
12     public static void main ( String[] argv ) {
13         Base x = new DerivedA();
14         DerivedA y = new DerivedA();
15         x.method();
16         y.method();
17         ( (DerivedA)x ).method();
```

Case Study I: Java

```
1  public class Lookup {
2      public static class Base {
3          public void method() {
4              System.out.println("Base.method");
5          }
6      }
7      public static class DerivedA extends Base {
8          public void method() {
9              System.out.println("DerivedA.method");
10         }
11     }
12     public static void main ( String[] argv ) {
13         Base x = new DerivedA();
14         DerivedA y = new DerivedA();
15         x.method();
16         y.method();
17         ( (DerivedA)x ).method();
18         ( (Base)y ).method();
19     }
20 }
```

Case Study I: C++

```
1  #include <stdio.h>
2
3  class Base {
4  public:
5     void method ( ) { printf("%s\n", __PRETTY_FUNCTION__); }
6  };
```

Case Study I: C++

```
1  #include <stdio.h>
2
3  class Base {
4  public:
5     void method ( ) { printf("%s\n", __PRETTY_FUNCTION__); }
6  };
7  class DerivedA : public Base {
8  public:
9     void method ( ) { printf("%s\n", __PRETTY_FUNCTION__); }
10 };
```

Case Study I: C++

```
1  #include <stdio.h>
2
3  class Base {
4  public:
5     void method ( ) { printf("%s\n", __PRETTY_FUNCTION__); }
6  };
7  class DerivedA : public Base {
8  public:
9     void method ( ) { printf("%s\n", __PRETTY_FUNCTION__); }
10 };
11 int main ( int argc, char **argv ) {
12    Base *x = new DerivedA();
13    DerivedA *y = new DerivedA();
```

Case Study I: C++

```
1  #include <stdio.h>
2
3  class Base {
4  public:
5     void method ( ) { printf("%s\n", __PRETTY_FUNCTION__); }
6  };
7  class DerivedA : public Base {
8  public:
9     void method ( ) { printf("%s\n", __PRETTY_FUNCTION__); }
10 };
11 int main ( int argc, char **argv ) {
12    Base *x = new DerivedA();
13    DerivedA *y = new DerivedA();
14    x->method();
```

Case Study I: C++

```
1  #include <stdio.h>
2
3  class Base {
4  public:
5     void method ( ) { printf("%s\n", __PRETTY_FUNCTION__); }
6  };
7  class DerivedA : public Base {
8  public:
9     void method ( ) { printf("%s\n", __PRETTY_FUNCTION__); }
10 };
11 int main ( int argc, char **argv ) {
12    Base *x = new DerivedA();
13    DerivedA *y = new DerivedA();
14    x->method();
15    y->method();
```

Case Study I: C++

```
1  #include <stdio.h>
2
3  class Base {
4  public:
5     void method ( ) { printf("%s\n", __PRETTY_FUNCTION__); }
6  };
7  class DerivedA : public Base {
8  public:
9     void method ( ) { printf("%s\n", __PRETTY_FUNCTION__); }
10 };
11 int main ( int argc, char **argv ) {
12    Base *x = new DerivedA();
13    DerivedA *y = new DerivedA();
14    x->method();
15    y->method();
16    ( (DerivedA*)x )->method();
```

Case Study I: C++

```
1  #include <stdio.h>
2
3  class Base {
4  public:
5     void method ( ) { printf("%s\n", __PRETTY_FUNCTION__); }
6  };
7  class DerivedA : public Base {
8  public:
9     void method ( ) { printf("%s\n", __PRETTY_FUNCTION__); }
10 };
11 int main ( int argc, char **argv ) {
12    Base *x = new DerivedA();
13    DerivedA *y = new DerivedA();
14    x->method();
15    y->method();
16    ( (DerivedA*)x )->method();
17    ( (Base*)y )->method();
18 }
```

Intermezzo: C++ with virtual methods

```
1  #include <stdio.h>
2
3  class A {
4  public:
5     virtual void method1 ( ) { printf("%s\n", __PRETTY_FUNCTION__); }
6     virtual void method2 ( ) { printf("%s\n", __PRETTY_FUNCTION__); }
7  };
```

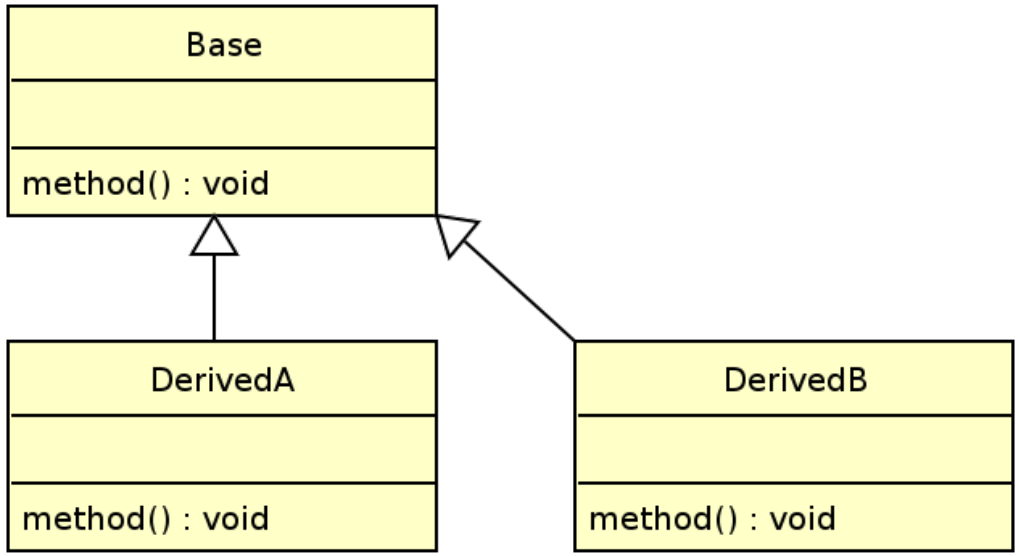
Intermezzo: C++ with virtual methods

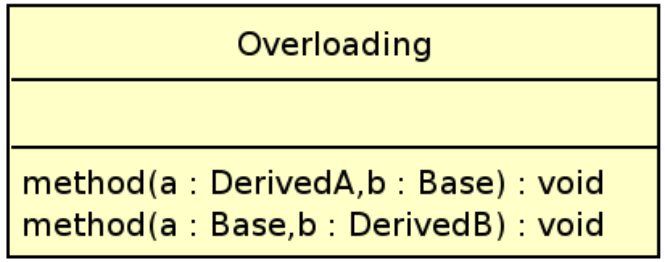
```
 1  #include <stdio.h>
 2
 3  class A {
 4  public:
 5     virtual void method1 ( ) { printf("%s\n", __PRETTY_FUNCTION__); }
 6     virtual void method2 ( ) { printf("%s\n", __PRETTY_FUNCTION__); }
 7  };
 8  class B {
 9  public:
10     virtual void method2 ( ) { printf("%s\n", __PRETTY_FUNCTION__); }
11     virtual void method1 ( ) { printf("%s\n", __PRETTY_FUNCTION__); }
12  };
```

Intermezzo: C++ with virtual methods

```
1  #include <stdio.h>
2
3  class A {
4  public:
5     virtual void method1 ( ) { printf("%s\n", __PRETTY_FUNCTION__); }
6     virtual void method2 ( ) { printf("%s\n", __PRETTY_FUNCTION__); }
7  };
8  class B {
9  public:
10    virtual void method2 ( ) { printf("%s\n", __PRETTY_FUNCTION__); }
11    virtual void method1 ( ) { printf("%s\n", __PRETTY_FUNCTION__); }
12 };
13 int main ( int argc, char **argv ) {
14    A *x = (A*)(new B());
15    x->method1();
16 }
```

Case Study II: Method Overloading






```
Overloading.method(new DerivedA(), DerivedB());
```

Case Study II: Java

```
1  public class Overloading {
2      public static class Base {}
3      public static class DerivedA extends Base {}
4      public static class DerivedB extends Base {}
5
```

Case Study II: Java

```
1  public class Overloading {
2      public static class Base {}
3      public static class DerivedA extends Base {}
4      public static class DerivedB extends Base {}
5
6      public static void method ( DerivedA a, Base b )
7          { System.out.println("method ( DerivedA a, Base b )"); }
```

Case Study II: Java

```
1  public class Overloading {
2      public static class Base {}
3      public static class DerivedA extends Base {}
4      public static class DerivedB extends Base {}
5
6      public static void method ( DerivedA a, Base b )
7          { System.out.println("method ( DerivedA a, Base b )"); }
8      public static void method ( Base a, DerivedB b )
9          { System.out.println("method ( Base a, DerivedB b )"); }
```

Case Study II: Java

```
1  public class Overloading {
2      public static class Base {}
3      public static class DerivedA extends Base {}
4      public static class DerivedB extends Base {}
5
6      public static void method ( DerivedA a, Base b )
7          { System.out.println("method ( DerivedA a, Base b )"); }
8      public static void method ( Base a, DerivedB b )
9          { System.out.println("method ( Base a, DerivedB b )"); }
10     public static void method ( Base a, Base b )
11         { System.out.println("method ( Base a, Base b )"); }
12
```

Case Study II: Java

```
1  public class Overloading {
2      public static class Base {}
3      public static class DerivedA extends Base {}
4      public static class DerivedB extends Base {}
5
6      public static void method ( DerivedA a, Base b )
7          { System.out.println("method ( DerivedA a, Base b )"); }
8      public static void method ( Base a, DerivedB b )
9          { System.out.println("method ( Base a, DerivedB b )"); }
10     public static void method ( Base a, Base b )
11         { System.out.println("method ( Base a, Base b )"); }
12
13     public static void main ( String[] args ) {
14         method (new DerivedA(), new DerivedB() );
```

Case Study II: Java

```
1  public class Overloading {
2      public static class Base {}
3      public static class DerivedA extends Base {}
4      public static class DerivedB extends Base {}
5
6      public static void method ( DerivedA a, Base b )
7          { System.out.println("method ( DerivedA a, Base b )"); }
8      public static void method ( Base a, DerivedB b )
9          { System.out.println("method ( Base a, DerivedB b )"); }
10     public static void method ( Base a, Base b )
11         { System.out.println("method ( Base a, Base b )"); }
12
13     public static void main ( String[] args ) {
14         method (new DerivedA(), new DerivedB() );
15         method ( (Base)(new DerivedA()) , (Base)(new DerivedB()) );
16     }
17 }
```

Case Study II: C++

```
1  #include <stdio.h>
2  class Base {};
3  class DerivedA : public Base {};
4  class DerivedB : public Base {};
```

Case Study II: C++

```
1  #include <stdio.h>
2  class Base {};
3  class DerivedA : public Base {};
4  class DerivedB : public Base {};
5
6  void method ( DerivedA *a, Base *b )
7  { printf("%s", __PRETTY_FUNCTION__); }
```

Case Study II: C++

```
1  #include <stdio.h>
2  class Base {};
3  class DerivedA : public Base {};
4  class DerivedB : public Base {};
5
6  void method ( DerivedA *a, Base *b )
7  { printf("%s", __PRETTY_FUNCTION__); }
8  void method ( Base *a, DerivedB *b )
9  { printf("%s", __PRETTY_FUNCTION__); }
```

Case Study II: C++

```
1  #include <stdio.h>
2  class Base {};
3  class DerivedA : public Base {};
4  class DerivedB : public Base {};
5
6  void method ( DerivedA *a, Base *b )
7  { printf("%s", __PRETTY_FUNCTION__); }
8  void method ( Base *a, DerivedB *b )
9  { printf("%s", __PRETTY_FUNCTION__); }
10 void method ( Base *a, Base *b )
11 { printf("%s", __PRETTY_FUNCTION__); }
```

Case Study II: C++

```
1  #include <stdio.h>
2  class Base {};
3  class DerivedA : public Base {};
4  class DerivedB : public Base {};
5
6  void method ( DerivedA *a, Base *b )
7  { printf("%s", __PRETTY_FUNCTION__); }
8  void method ( Base *a, DerivedB *b )
9  { printf("%s", __PRETTY_FUNCTION__); }
10 void method ( Base *a, Base *b )
11 { printf("%s", __PRETTY_FUNCTION__); }
12
13 int main ( int argc, char **argv ) {
14    method( new DerivedA(), new DerivedB() );
```

Case Study II: C++

```
1  #include <stdio.h>
2  class Base {};
3  class DerivedA : public Base {};
4  class DerivedB : public Base {};
5
6  void method ( DerivedA *a, Base *b )
7  { printf("%s", __PRETTY_FUNCTION__); }
8  void method ( Base *a, DerivedB *b )
9  { printf("%s", __PRETTY_FUNCTION__); }
10 void method ( Base *a, Base *b )
11 { printf("%s", __PRETTY_FUNCTION__); }
12
13 int main ( int argc, char **argv ) {
14    method( new DerivedA(), new DerivedB() );
15    method( (Base*)(new DerivedA()), (Base*)(new DerivedB()) );
16 }
```

Case Study II: CLOS

```
1 (defclass Base () () )
2 (defclass DerivedA ( Base ) () )
3 (defclass DerivedB ( Base ) () )
```

Case Study II: CLOS

```
1  (defclass Base () () )
2  (defclass DerivedA ( Base ) () )
3  (defclass DerivedB ( Base ) () )
4
5  (defmethod method ( (a DerivedA) (b Base) )
6    (format t "(a DerivedA) (b Base)"))
```

Case Study II: CLOS

```
1  (defclass Base () () )
2  (defclass DerivedA ( Base ) () )
3  (defclass DerivedB ( Base ) () )
4
5  (defmethod method ( (a DerivedA) (b Base) )
6    (format t "(a DerivedA) (b Base)"))
7  (defmethod method ( (a Base) (b DerivedB) )
8    (format t "(a Base) (b DerivedB)") )
```

Case Study II: CLOS

```
1  (defclass Base () () )
2  (defclass DerivedA ( Base ) () )
3  (defclass DerivedB ( Base ) () )
4
5  (defmethod method ( (a DerivedA) (b Base) )
6    (format t "(a DerivedA) (b Base)"))
7  (defmethod method ( (a Base) (b DerivedB) )
8    (format t "(a Base) (b DerivedB)") )
9  (defmethod method ( (a Base) (b Base) )
10   (format t "(a Base) (b Base)") )
```

Case Study II: CLOS

```
1  (defclass Base () () )
2  (defclass DerivedA ( Base ) () )
3  (defclass DerivedB ( Base ) () )
4
5  (defmethod method ( (a DerivedA) (b Base) )
6    (format t "(a DerivedA) (b Base)"))
7  (defmethod method ( (a Base) (b DerivedB) )
8    (format t "(a Base) (b DerivedB)") )
9  (defmethod method ( (a Base) (b Base) )
10   (format t "(a Base) (b Base)") )
11 (method
12  (make-instance 'DerivedA)
13  (make-instance 'DerivedB))
```

Outline

Message send

X

Method invocation

Message send

X

Method invocation

Method lookup algorithm

Input

receiver

method signature

Output

method

Early binding

In compile-time

No memory overhead

No performance overhead

No polymorphism

C++

Late binding

In run-time

Memory overhead

Performance overhead

Polymorphism

Smalltalk, Java, Self, Objective C. . .

Pseudo-Late binding

In run-time

Memory overhead

Less performance overhead than late binding

Polymorphism only within a class hierarchy

C++, Object pascal

Method Lookup & Overloading

Java, C++

- Compile-time
- Different signatures

Method Lookup & Overloading

Java, C++

- Compile-time
- Different signatures

CLOS

- Run-time
- Same signatures

Method lookup in Java...

```
1  lookup-method-in-class ( class , selector )

2  lookup-method ( receiver, selector ) {
3    currentClass←class-of ( receiver )
4    while ( currentClass ≠ nil ) {
5      method←lookup-method-in-class
6                  ( currentClass , selector )
7      if ( method ≠ nil )
8        ↑method
9      currentClass←superclass-of ( currentClass )
10   }
11   throw MethodNotFoundException
12 }
```

Method lookup in Smalltalk...

```
1  lookup-method-in-class ( class , selector )

2  lookup-method ( receiver, selector ) {
3    currentClass←class-of ( receiver )
4    while ( currentClass ≠ nil ) {
5      method←lookup-method-in-class
6                  ( currentClass , selector )
7      if ( method ≠ nil )
8        ↑method
9      currentClass←superclass-of ( currentClass )
10   }
11   if ( selector = #doesNotUnderstand:)
12     error ( "Recursive does not understand" )
13   ↑lookup-method ( receiver , #doesNotUnderstand:)
14 }
```

Outline

- Is pseudo-late binding slower than early binding?

- Is pseudo-late binding slower than early binding?
  Yes. . .

- Is pseudo-late binding slower than early binding?
  Yes. . .
- Is true late-binding slower than pseudo-late binding?

- Is pseudo-late binding slower than early binding?
  Yes. . .
- Is true late-binding slower than pseudo-late binding?
  No. Well, not always. . .

- Is pseudo-late binding slower than early binding?
  Yes. . .
- Is true late-binding slower than pseudo-late binding?
  No. Well, not always. . .
- Is late-binding better?

- Is pseudo-late binding slower than early binding?
  Yes. . .
- Is true late-binding slower than pseudo-late binding?
  No. Well, not always. . .
- Is late-binding better?
  Definitely! :-)

Outline

Problem

Same operation, different algorithms for different argument types

Example

Summing method for integers, floats & fractions

Solution in Java

```
1  public class SmallInteger extends Number
2    public Number add ( SmallInteger i ) {
3      ...sum SmallInteger and SmallInteger ...
4    }
5    public Number add ( Float f ) {
6      ...sum SmallInteger and Float ...
7    }
8    public Number add ( Fraction f ) {
9      ...sum SmallInteger and Fraction ...
10   }
```

Solution in Smalltalk

```
1 SmallInteger ≫+ anObject
2   (anObject isKindOf:  SmallInteger) ifTrue:
3     [ ...sum SmallInteger and SmallInteger...].
4   (anObject isKindOf:  Float) ifTrue:
5     [ ...sum SmallInteger and Float...].
6   (anObject isKindOf:  Fraction) ifTrue:
7     [ ...sum SmallInteger and Fraction...].
```

More Smalltalkish Solution

```
1  SmallInteger ≫+ anObject
2    ↑anObject sumFromSmallInteger:  self
3
4  SmallInteger ≫sumFromSmallInteger:  anObject
5    ↑...sum SmallInteger and SmallInteger...
6
7  Float ≫sumFromSmallInteger:  anObject
8    ↑...sum Float and SmallInteger...
9
10 Fraction ≫sumFromSmallInteger:  anObject
11   ↑...sum Fraction and SmallInteger...
```