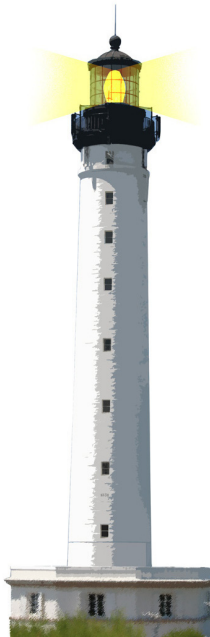
About type and method lookup



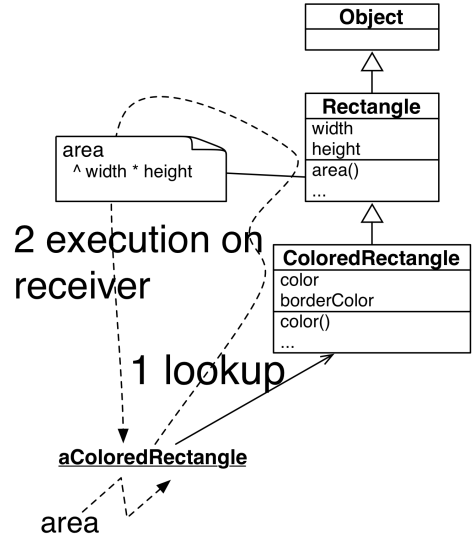
Outline

- Lookup (remember)
- Static type vs Dynamic type
- Overloading and types
- Method lookup

Message Sending

Sending a **message** is a two-step process:

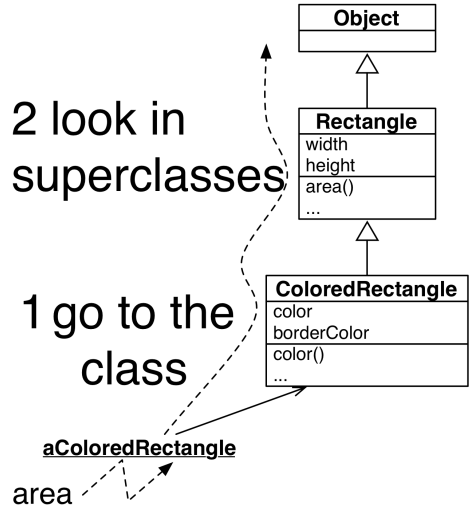
1. **look up** the **method** matching the message
2. execute this method on the **receiver**



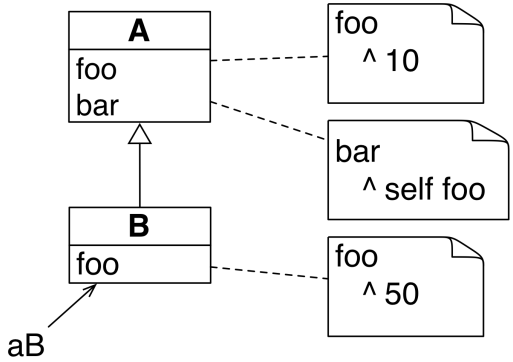
Method lookup

The lookup starts in the **class** of the **receiver** then:

- if the method is defined in the class, it is returned
- otherwise the search continues in the superclass

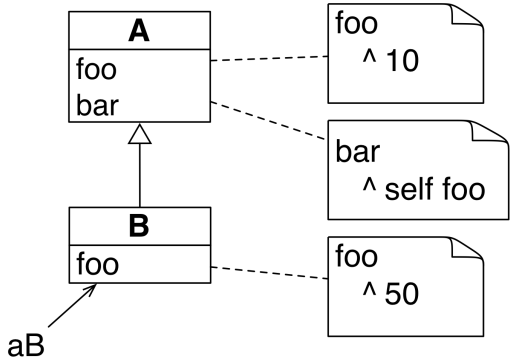


self/this always represents the receiver




```
A new foo
> ...
B new foo
> ...
```

self/this always represents the receiver




```
A new foo
> 10
B new foo
> 50
```

It was the essence

Questions:

- How types influence (polute) this beautiful model?
- Static types, dynamic types, overloading

Static type

Consider:

```
A a = new B();
```

- The static type of variable `a` is `A` i.e., the statically declared class to which it refers.
- The static type never changes.

Dynamic type

Consider:

```
A a = new B();
```

- The dynamic type of a is B i.e., the class of the object currently bound to a.
- The dynamic type may change throughout the program.

```
a = new A();
```

Now the dynamic type is also A!

Static type

Pay attention method signatures also define static types

```
foo (A a){

}

foo(new B());
```

static type of a is A, dynamic type of a is B

Overloading

How are overloaded method calls resolved?

```
class A { }
class B extends A { }
void m(A a1, A a2) { println("m(A,A)"); };
void m(A a1, B b1) { println("m(A,B)"); };
void m(B b1, A a1) { println("m(B,A)"); };
void m(B b1, B b2) { println("m(B,B)"); };
```

```
B b = new B();
A a = b;
```

a and b have a dynamic type B

How are overloaded method calls resolved?

```
class A { }
class B extends A { }
void m(A a1, A a2) { println("m(A,A)"); };
void m(A a1, B b1) { println("m(A,B)"); };
void m(B b1, A a1) { println("m(B,A)"); };
void m(B b1, B b2) { println("m(B,B)"); };
```

```
B b = new B();
A a = b;
```

Which is considered: the static or dynamic argument type?

```
m(a, a);
m(a, b);
m(b, a);
m(b, b);
```

How are overloaded method calls resolved?

```
class A { }
class B extends A { }
void m(A a1, A a2) { println("m(A,A)"); };
void m(A a1, B b1) { println("m(A,B)"); };
void m(B b1, A a1) { println("m(B,A)"); };
void m(B b1, B b2) { println("m(B,B)"); };
```

```
B b = new B();
A a = b;
```

Which is considered: the static or dynamic argument type?

```
m(a, a);   m(A,A)
m(a, b);   m(A,B)
m(b, a);   m(B,A)
m(b, b);   m(B,B)
```

The static type of arguments is always used to resolve overloaded method calls.

Overloaded method calls

The static type of argument is always used

- no dynamic dispatch
- force you to cast
- force you to use `getClass`

Avoid overloading (See the Lecture on Overloading)

How do static and dynamic types interact?

```
class A {
  void m(A a) { println("A.m(A)"); }}
class B extends A {
  void m(B b) { println("B.m(B)"); }}
```

```
B b = new B(); A a = b;
```

What are the results of the invocations?

```
a.m(a);
a.m(b);
b.m(a);
b.m(b);
```

How do static and dynamic types interact?

```
class A {
  void m(A a) { println("A.m(A)"); }}
class B extends A {
  void m(B b) { println("B.m(B)"); }}
```

```
B b = new B(); A a = b;
```

What are the results of the invocations?

```
a.m(a);  A.m(A)
a.m(b);  A.m(A)
b.m(a);  A.m(A)
b.m(b);  B.m(B)
```

- Static types determine which message is sent.
- Dynamic types determine which method is called.

Compilation vs. execution

At compilation:

- First, the static type of the receiver determines which class we consider
- Second, does the class define the method?
- Third, does the static type of the arguments fit the static type of the parameter?
- Fourth, find the best fit

At execution:

- the lookup starts in the class of the receiver

a.m(a)

```
class A {void m(A a) { println("A.m(A)"); }}
class B extends A {void m(B b) { println("B.m(B)"); }}
B b = new B(); A a = b;
```

- Step 1: receiver static type is A: we look in A
- Step 2: there is a method m
- Step 3: static type of a matches A a we will be looking for "m(A a)"

The dynamic type of a is B. The lookup starts in class B but looks for "m(A a)" > A.m(A)

b.m(a)

```
class A {void m(A a) { println("A.m(A)"); }}
class B extends A {void m(B b) { println("B.m(B)"); }}
B b = new B(); A a = b;
```

- Step 1: the static type of b is B, so we look in B and its superclass A
- Step 2: There is a method m (in fact two m(A a) and m(B b)
- Step 3: the static type of a is A we will be looking for m(A a)

The dynamic type of b is B.

- The lookup starts in class B and looks for m(A a)

> A.m(A)

b.m(b)

```
class A {void m(A a) { println("A.m(A)"); }}
class B extends A {void m(B b) { println("B.m(B)"); }}
B b = new B(); A a = b;
```

- Step 1: b static type is B, so we look in B and its superclass A
- Step 2: There is a method m (in fact two m(A a) and m(B b)
- Step 3: the static type of b is B we will be looking for m(B b)""

the lookup starts in class B and looks for m(B b) > B.m(B)

a.m(b)

```
class A {void m(A a) { println("A.m(A)"); }}
class B extends A {void m(B b) { println("B.m(B)"); }}
B b = new B(); A a = b;
```

- Step 1: receiver static type is A: we only look in A
- Step 2: there is a method m
- Step 3: the static type of b is B but since A is a supertype of B this is ok we will be looking for m(A a)

The dynamic type of a is B

- The lookup starts in class B and looks for m(A a)

> A.m(A)

a.m(c)

```
class A {void m(A a) { println("A.m(A)"); }}
class B extends A {void m(B b) { println("B.m(B)"); }}
B b = new B(); A a = b; C c = new C;
```

- Step 1: We look only in A
- Step 2: there is a method m
- Step 3: C the static type of c does not match A there is no subtype relations

Does not compile!

Conclusion

- Avoid overloading as much as possible (it is cool 2min and painful all the rest of the time)
- Avoid direct class in fields and signature
  - Better use interfaces

A course by

S. Ducasse, G. Polito, and Pablo Tesone