About type and method lookup

S. Ducasse, L. Fabresse, G. Polito, and P. Tesone
Outline

- Lookup (remember)
- Static type vs Dynamic type
- Type checker
- 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
It was the essence

Questions:

- How do types influence (pollute) this beautiful model?
- Static types, dynamic types, overloading?
The type of a variable

Let’s take a simple program model:

- a variable is a box with a label: its type.
- a variable contains a reference to objects.

A variable type indicates the kind of object the variable can refer to.
A a: we can put reference to objects of the class A (and subclasses)
Type checker

During compilation

- A type checker is a tool that tries to make sure that correct objects are put in variables
- Using type information the type checker avoids an unknown message being sent to an object
Static vs. Dynamic Types

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

- The static type of variable `a` is `A` i.e. the declared label of the box.
  - The static type never changes.
- 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.

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

Now the dynamic type is also `A`!
Static and dynamic types can be different

Consider:

A a = new B();

- The static type of variable a is A.
- The dynamic type of a is B.
Pay attention to method signatures also define static types

```java
foo (A a){
}
foo(new B());
```

the static type of a is A, dynamic type of a is B
How do static and dynamic types interact?

```java
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?

```shell
a.m(a);
a.m(b);
b.m(a);
b.m(b);
```
How do static and dynamic types interact?

```java
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
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 look 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)
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 look for `m(A a)`

The dynamic type of `b` is `B`.
- The lookup starts in class `B` and looks for `m(A a)`
- `b.m(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 look for m(B b)
- The lookup starts in class B and looks for m(B b)
- > 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: 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 look 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)
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

- Examples used so far were simple
- Be careful with static types, it can get tricky
- More details on the lectures on overloading if needed
- More details on the lectures on interfaces if needed