

CS 213 : Software Methodology

Spring 2017

Sesh Venugopal

Lecture 7: Feb 7

Inheritance: Object class/equals method – Part 2

Background: Method Overloading/Overriding

Method **Overloading**:

Two methods in a class have the same name but different numbers, types, or sequences of parameters

```
class Test {  
    int m(int x) {...}  
    int m(float y) {...}  
}
```

Overloaded method m

```
class Test {  
    int m(int x) {...}  
    float m(float y) {...}  
}
```

Overloaded method m

```
class Test {  
    int m(int x) {...}  
    float m(int y) {...}  
}
```

Error

Two or more methods in a class are **overloaded** if they have the same name but different signatures

signature = name + params (return type NOT included in signature)

Method **Overriding**:

A method in a subclass has the same signature as in the superclass

equals overload/override

```
public class Point {
    int x,y;
    .
    .
    .
    public boolean equals(Object o) {
        if (o == null ||
            (!(o instanceof Point))) {
            return false;
        }
        Point other = (Point)o;
        return x == other.x &&
            y == other.y
    }

    public boolean equals(Point p) {
        if (p == null) {
            return false;
        }
        return x == p.x && y == p.y
    }
}
```

With the following setup:

```
Object o = new Object();
```

```
Point p = new Point(3,4);
```

```
Object op = new Point(3,4);
```

Which method is called in each case,
and what's the result of the call?:

`p.equals(o);` // ? **False**

`p.equals(p);` // ? **True**

`p.equals(op);` // ? **True**

equals overload/override

```
public class Point {  
    int x,y;  
    .  
    .  
    .  
    public boolean equals(Object o) {  
        if (o == null ||  
            (!(o instanceof Point))) {  
            return false;  
        }  
        Point other = (Point)o;  
        return x == other.x &&  
            y == other.y  
    }  
  
    public boolean equals(Point p) {  
        if (p == null) {  
            return false;  
        }  
        return x == p.x && y == p.y  
    }  
}
```

With the following setup:

```
Object o = new Object();
```

```
Point p = new Point(3,4);
```

```
Object op = new Point(3,4);
```

Which method is called in each case,
and what's the result of the call?:

```
op.equals(o); // ? False
```

```
op.equals(p); // ? True
```

```
op.equals(op); // ? True
```

Method Overloading/Overriding

Static and Dynamic Types

What rules determine which method is called?

A. First, the **COMPILER** determines the *signature* of the method that will be called:

1. Look at the static type of the object (“target”) on which method is called.
Say this type/class is X

```
Object o = new Object();  
Point p = new Point(3,4);  
Object op = new Point(3,4);
```

```
p.equals(o);
```

```
p.equals(p);
```

```
p.equals(op);
```

Static type of
p is Point

```
op.equals(o);
```

```
op.equals(p);
```

```
op.equals(op);
```

Static type of
op is Object

Method Overloading/Overriding

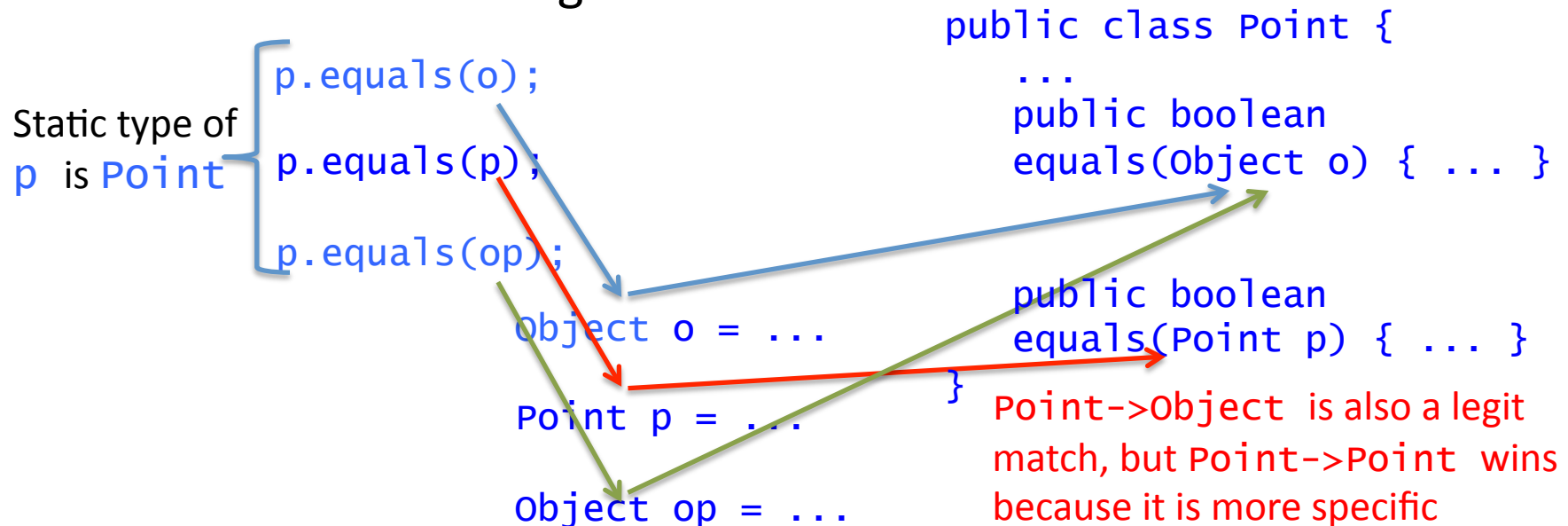
Static and Dynamic Types

What rules determine which method is called?

A. First, the **COMPILER** determines the *signature* of the method that will be called:

2. In the class X, find a method whose name matches the called method, and whose parameters most specifically match the static types of the arguments at call

e.g. X is **Point**



Method Overloading/Overriding

Static and Dynamic Types

What rules determine which method is called?

A. First, the **COMPILER** determines the *signature* of the method that will be called:

2. In the class X, find a method whose name matches the called method, and whose parameters most specifically match the static types of the arguments at call

e.g. X is **Object**

```
op.equals(o);  
op.equals(p);  
op.equals(op);
```

Static type of
op is **Object**

Object has a single **equals**
method that matches all
of these calls

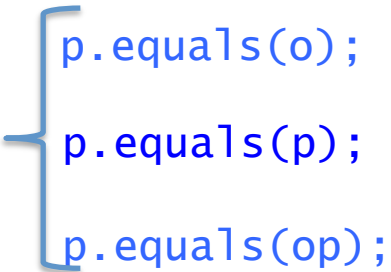
Method Overloading/Overriding

Static and Dynamic Types

What rules determine which method is called?

B. At run time, the runtime/actual “target” (called) object, or its superclass chain is searched for the determined signature, and the matching method executed

Static type of `p` is `Point`



```
p.equals(o);  
p.equals(p);  
p.equals(op);
```

`Point` defines `equals(Object)` as well as `equals(Point)`, which match with the respective statically bound method signatures

```
Point p = new Point(3,4);
```

Dynamic type of `p` is `Point`,

Method Overloading/Overriding

Static and Dynamic Types

What rules determine which method is called?

B. At run time, the runtime/actual “target” (called) object, or its superclass chain is searched for the determined signature, and the matching method executed

What if `Point` did NOT override `equals(Object)` inherited from `Object`?

Static type of `p` is `Point`

- `p.equals(o);` → Would call `Object` version of `equals`
superclass `Object` has `equals(Object)`
- `p.equals(p);`
- `p.equals(op);` → Would call `Object` version of `equals`
Bad news: result is false, even though but objects are (3,4)!!

`Point p = new Point(3,4);`

Dynamic type of `p` is `Point`,

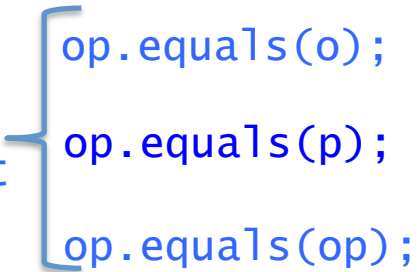
Method Overloading/Overriding

Static and Dynamic Types

What rules determine which method is called?

B. At run time, the runtime/actual “target” (called) object, or its superclass chain is searched for the determined signature, and the matching method executed

Static type of
`op` is `Object`



```
op.equals(o);  
op.equals(p);  
op.equals(op);
```

`Point` defines `equals(Object)`
which matches with the statically
bound methods

```
Object op = new Point(3,4);
```

Dynamic type of `op` is `Point`

Method Overloading/Overriding

Static and Dynamic Types

What rules determine which method is called?


B. At run time, the runtime/actual “target” (called) object, or its superclass chain is searched for the determined signature, and the matching method executed

What if `Point` did NOT override `equals(Object)` inherited from `Object`?

```
Object op = new Point(3,4);
```

Dynamic type of `op` is `Point`

Static type of
`op` is `Object`



```
op.equals(o); // ? False  
op.equals(p); // ? False  
op.equals(op); // ? True
```

Bad news: result is false, even though both objects are (3,4)!!

All these calls would be to the
`Object` version of `equals`

Method Overloading/Overriding

Static and Dynamic Types

What if the inherited `equals(Object)` is not overridden, and only `equals(Point)` is coded?

The call `op.equals(p)` will result in false, which fails the requirement of (3,4) being equal to (3,4), even if the point objects are physically different

So, the inherited `equals(Object)` must be overridden

Is it sufficient to only override the inherited `equals(Object)`, and not code an `equals(Point)` method?

Yes

Is it detrimental/inadvisable to have both?

Yes, it leads to avoidable confusion, so removing `equals(Point)` is clearer/unambiguous/better design