

# Overloading/Overriding/Hiding

George Fourtounis

Dept. of Informatics and Telecommunications  
University of Athens

# Overloading

```
class Point {  
    float x, y;  
    void move(int dx, int dy) {  
        x += dx;  
        y += dy;  
    }  
    void move(float dx, float dy) {  
        x += dx;  
        y += dy;  
    }  
}
```

```
Point p = new Point();
```

(“Example 8.4.9-1” from the Java 8 Language Specification, slightly modified.)

- `p.move(10.1f, 3.0f)?`
- `p.move(10, 10)?`
- choose the *most specific* method

# Overriding (and super)

```
class Point {
    int x = 0, y = 0;
    void move(int dx, int dy) { x += dx; y += dy; }
}

class SluggishPoint extends Point {
    void move(int dx, int dy) { x += dx / 2; y += dy / 2; }
}

class SlowPoint extends Point {
    int xLimit, yLimit;
    void move(int dx, int dy) {
        super.move(limit(dx, xLimit), limit(dy, yLimit));
    }
    static int limit(int d, int limit) {
        return d > limit ? limit : d < -limit ? -limit : d;
    }
}
```

(Example 8.4.8.1-1 from the Java 8 Language Specification, modified.)

# Shadowing

- Shadowing an instance variable:

```
class Test {  
    static int x = 1;  
    public static void main(String[] args) {  
        int x = 0;  
        System.out.print("x=" + x);  
        System.out.println(", Test.x=" + Test.x);  
    }  
}
```

- Parameter names and local variables cannot shadow each other.
- Duplicate definitions of locals are an error:

```
class Test1 {  
    public static void main(String[] args) {  
        int i;  
        for (int i = 0; i < 10; i++)    // ERROR!  
            System.out.println(i);  
    }  
}
```

(Examples 6.4.1-1 and 6.4-1 from the Java 8 Language Specification.)

# Shadowing

- Shadowing an instance variable:

```
class Test {  
    static int x = 1;  
    public static void main(String[] args) {  
        int x = 0;  
        System.out.print("x=" + x);  
        System.out.println(", Test.x=" + Test.x);  
    }  
}  
x=0, Test.x=1
```

- Parameter names and local variables cannot shadow each other.
- Duplicate definitions of locals are an error:

```
class Test1 {  
    public static void main(String[] args) {  
        int i;  
        for (int i = 0; i < 10; i++)    // ERROR!  
            System.out.println(i);  
    }  
}
```

(Examples 6.4.1-1 and 6.4-1 from the Java 8 Language Specification.)

# Hiding Fields

```
class A {  
    int x = 1;  
    void m() { System.out.println("A.x = "+x); }  
}
```

```
class B extends A {  
    /* This could also be 'static'. */  
    String x = "hello";  
    void k() { System.out.println("B.x = "+x); }  
}
```

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

• a.m();

# Hiding Fields

```
class A {  
    int x = 1;  
    void m() { System.out.println("A.x = "+x); }  
}
```

```
class B extends A {  
    /* This could also be 'static'. */  
    String x = "hello";  
    void k() { System.out.println("B.x = "+x); }  
}
```

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

• `a.m();`       $A.x = 1$

# Hiding Fields

```
class A {  
    int x = 1;  
    void m() { System.out.println("A.x = "+x); }  
}
```

```
class B extends A {  
    /* This could also be 'static'. */  
    String x = "hello";  
    void k() { System.out.println("B.x = "+x); }  
}
```

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

• a.m();            A.x = 1

• b.m();



# Hiding Fields

```
class A {  
    int x = 1;  
    void m() { System.out.println("A.x = "+x); }  
}
```

```
class B extends A {  
    /* This could also be 'static'. */  
    String x = "hello";  
    void k() { System.out.println("B.x = "+x); }  
}
```

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

• a.m();            A.x = 1

• b.m();            A.x = 1

# Hiding Fields

```
class A {  
    int x = 1;  
    void m() { System.out.println("A.x = "+x); }  
}
```

```
class B extends A {  
    /* This could also be 'static'. */  
    String x = "hello";  
    void k() { System.out.println("B.x = "+x); }  
}
```

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

• a.m();            A.x = 1

• b.m();            A.x = 1

• b.k();

# Hiding Fields

```
class A {  
    int x = 1;  
    void m() { System.out.println("A.x = "+x); }  
}
```

```
class B extends A {  
    /* This could also be 'static'. */  
    String x = "hello";  
    void k() { System.out.println("B.x = "+x); }  
}
```

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

- `a.m();`            *A.x = 1*
- `b.m();`            *A.x = 1*
- `b.k();`            *B.x = hello*

# Hiding Methods

Re-definition of `static` method also hides it in the superclass. A static method cannot hide a non-static method. Hidden methods can still be accessed with:

- `super.m()`
- `((SuperClass)obj).m()`
- `Fully.Qualified.Name.m()`

# Overriding, Overloading, and Hiding

```
class Point {
    int x = 0, y = 0;
    void move(int dx, int dy) {
        x += dx;
        y += dy;
    }
    int color;
}

class RealPoint extends Point {
    float x = 0.0f, y = 0.0f;
    void move(int dx, int dy) {
        move((float)dx, (float)dy);
    }
    void move(float dx, float dy) {
        x += dx;
        y += dy;
    }
}
```

(Example 8.4.9-2 from the Java 8 Language Specification.)

# Overriding (Wrong)

```
class Point {
    int x = 0, y = 0;
    void move(int dx, int dy) { x += dx; y += dy; }
    int getX() { return x; }
    int getY() { return y; }
    int color;
}

class RealPoint extends Point {
    float x = 0.0f, y = 0.0f;
    void move(int dx, int dy) { move((float)dx, (float)dy); }
    void move(float dx, float dy) { x += dx; y += dy; }
    float getX() { return x; }
    float getY() { return y; }
}
```

```
error: getX() in RealPoint cannot override getX() in Point
float getX() { return x; }
~
```

return type float is not compatible with int

```
error: getY() in RealPoint cannot override getY() in Point
float getY() { return y; }
~
```

return type float is not compatible with int

2 errors

# Overriding (Correct)

```
class Point {
    int x = 0, y = 0;
    void move(int dx, int dy) { x += dx; y += dy; }
    int getX() { return x; }
    int getY() { return y; }
    int color;
}

class RealPoint extends Point {
    float x = 0.0f, y = 0.0f;
    void move(int dx, int dy) { move((float)dx, (float)dy); }
    void move(float dx, float dy) { x += dx; y += dy; }
    int getX() { return (int)Math.floor(x); }
    int getY() { return (int)Math.floor(y); }
}
```

(Example 8.4.9-2 from the Java 8 Language Specification.)

# C++

```
class A {  
public:  
  
    void f()          { cout << "A::f() was called."      << endl; }  
    void f(int i) { cout << "A::f(int) was called." << endl; }  
};  
class B : public A {  
public:  
  
    void f()          { cout << "B::f() was called."      << endl; }  
};  
A a;  
B b;  
A* a2 = &b;
```

```
a.f();  
a.f(1);  
b.f();  
b.f(1);  
a2->f();
```



# C++

```
class A {  
public:  
  
    void f()          { cout << "A::f() was called."      << endl; }  
    void f(int i) { cout << "A::f(int) was called." << endl; }  
};  
class B : public A {  
public:  
  
    void f()          { cout << "B::f() was called."      << endl; }  
};  
A a;  
B b;  
A* a2 = &b;
```

```
a.f();      A::f() was called.  
a.f(1);  
b.f();  
b.f(1);  
a2->f();
```

# C++

```
class A {  
public:  
  
    void f()          { cout << "A::f() was called."      << endl; }  
    void f(int i) { cout << "A::f(int) was called." << endl; }  
};  
class B : public A {  
public:  
  
    void f()          { cout << "B::f() was called."      << endl; }  
};  
A a;  
B b;  
A* a2 = &b;
```

```
a.f();      A::f() was called.  
a.f(1);     A::f(int) was called.  
b.f();  
b.f(1);  
a2->f();
```

# C++

```
class A {  
public:  
  
    void f()          { cout << "A::f() was called."      << endl; }  
    void f(int i) { cout << "A::f(int) was called." << endl; }  
};  
class B : public A {  
public:  
  
    void f()          { cout << "B::f() was called."      << endl; }  
};  
A a;  
B b;  
A* a2 = &b;
```

```
a.f();      A::f() was called.  
a.f(1);     A::f(int) was called.  
b.f();      B::f() was called.  
b.f(1);  
a2->f();
```

# C++

```
class A {  
public:  
  
    void f()          { cout << "A::f() was called."      << endl; }  
    void f(int i) { cout << "A::f(int) was called." << endl; }  
};  
class B : public A {  
public:  
    using A::f;  
    void f()          { cout << "B::f() was called."      << endl; }  
};  
A a;  
B b;  
A* a2 = &b;
```

a.f();	<i>A::f() was called.</i>
a.f(1);	<i>A::f(int) was called.</i>
b.f();	<i>B::f() was called.</i>
b.f(1);	<i>A::f(int) was called. (needs using A::f in B)</i>
a2->f();	

# C++

```
class A {  
public:  
  
    void f()          { cout << "A::f() was called."      << endl; }  
    void f(int i) { cout << "A::f(int) was called." << endl; }  
};  
class B : public A {  
public:  
  
    void f()          { cout << "B::f() was called."      << endl; }  
};  
A a;  
B b;  
A* a2 = &b;
```

a.f();	<i>A::f() was called.</i>
a.f(1);	<i>A::f(int) was called.</i>
b.f();	<i>B::f() was called.</i>
b.f(1);	<i>A::f(int) was called. (needs using A::f in B)</i>
a2->f();	<i>A::f() was called. (no virtual)</i>

# C++

```
class A {  
public:  
    virtual  
    void f()          { cout << "A::f() was called."      << endl; }  
    void f(int i) { cout << "A::f(int) was called." << endl; }  
};  
class B : public A {  
public:  
  
    void f()          { cout << "B::f() was called."      << endl; }  
};  
A a;  
B b;  
A* a2 = &b;
```

a.f();	<i>A::f() was called.</i>
a.f(1);	<i>A::f(int) was called.</i>
b.f();	<i>B::f() was called.</i>
b.f(1);	<i>A::f(int) was called. (needs using A::f in B)</i>
a2->f();	<i>B::f() was called. (with virtual)</i>