Discussion 4: Inheritance

Administrivia

- Project 1B released, due Friday 2/9
- Midterm 2/12, 8-10 PM
  - Check Piazza for room assignments
  - Material up to 2/7
- Guerrilla Section this Saturday 2/10
- Sign up for CSM sections ASAP
- One-on-one tutoring sign ups on Piazza weekly

Subclass

Superclass

```
public class Fish {

     int weight;

     public Fish(int w) {
          weight = w;
     }

     public void swim() {
          System.out.println(“splash”);
     }
}
```

This invokes the super class’s constructor

```
public class Salmon extends Fish {

     String home;

     public Salmon(int w, String h) {
          super(w);
          home = h;
     }

     public void migrate() {
          System.out.println(“Migrating to ”
+ home);
     }

}
```

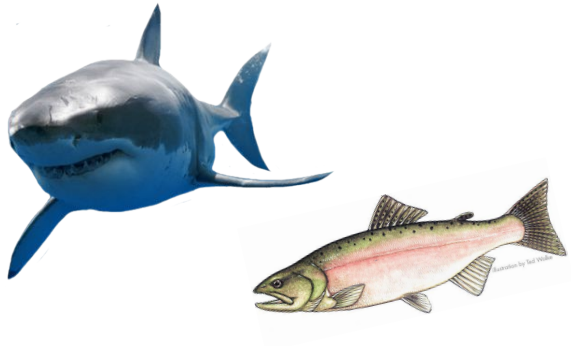
Salmon inherits these from Fish

Static vs Dynamic type: what fits in the box?

Static type

Dynamic type

```
Fish fish1 = new Fish();

Fish fish2 = new Salmon();

Salmon salmon1 = new Salmon();

Salmon salmon2 = new Fish();

Salmon salmon3 = fish2;
```



Overriding/Overloading

```
public class Fish {

     int weight;

     public void swim() {
          System.out.println(“splash”);
     }
}
```

Overriding: exact same method signature

Overloading: same name, different parameters

```
public class Salmon extends Fish {
     String home;

     public Salmon(int w, String h) {
          super(w);
          home = h;
     }

     public void swim() {
          System.out.println(“splish splash”);
     }

     public void swim(int speed) {
          System.out.println(“swimming at “ +
speed + “ mph”);
     }
}
```

Dynamic Method Selection

```
Fish fish = new Fish();
Salmon salmon = new Salmon();
Fish bob = new Salmon();

fish.swim();
salmon.swim();
bob.swim();

fish.swim(5);
salmon.swim(5);
bob.swim(5)
```

Compile time: static method lookup

Run time: dynamic method lookup

Dynamic Method Selection

Compile time: static method lookup

Run time: dynamic method lookup

```
Fish fish = new Fish();
Salmon salmon = new Salmon();
Fish bob = new Salmon();

fish.swim();                    splash
salmon.swim();                  splish splash
bob.swim();                     splish splash

fish.swim(5);                   compile-time error
salmon.swim(5);                 swimming at 5 mph
bob.swim(5)                     compile-time error
```

Casting

```
Fish redFish = new Fish();
Fish blueFish = new Salmon();

blueFish.swim(5); /* This wouldn’t compile before */

((Salmon) blueFish).swim(5); /* Now it does! */

((Salmon) redFish).swim(5); /* This compiles but gives you a
runtime error (ClassCastException) */
```

Interfaces

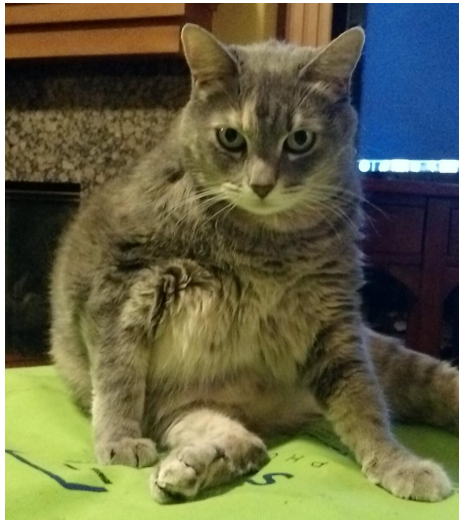
```
public interface Plant {
    public void grow();
    public void photosynthesize();
}

public class Rose implements Plant {
    public int height;
    public void grow() { height += 1; }
    public void photosynthesize() {
        System.out.println("Rose feels energized!");
    }
}
```

Rose promises to provide implementations of all of Plant's abstract methods

```
Plant plant = new Rose();
Plant plant = new Plant();
```

Creating Cats

- What will Cat inherit from Animal?
- How can we rewrite as little code as possible?



Creating Cats

```
public class Cat extends Animal {

    public Cat(String name, int age) {
        super(name, age);       // Call superclass' constructor.
        this.noise = "Meow!";   // Change the value of the field.
    }

    @Override
    public void greet() {
        System.out.println("Cat " + name + " says: " + makeNoise());
    }
}
```

Raining Cats and Dogs

- Remember dynamic method lookup for overridden methods
- Casting forces the compile-time type of an object

johnsu.deviantart.com/art/Raining-Cats-and-Dogs-30291521

Raining Cats and Dogs

```
Animal a = new Animal("Pluto", 10);
Cat c = new Cat("Garfield", 6);
Dog d = new Dog("Fido", 4);
a.greet();              // (A) Animal Pluto says: Huh?
c.greet();              // (B) Cat Garfield says: Meow!
d.greet();              // (C) Dog Fido says: WOOF!
a = c;
((Cat) a).greet();      // (D) Cat Garfield says: Meow!
a.greet();              // (E) Cat Garfield says: Meow!
```

What if we added this line?

```
Animal a = new Animal("Pluto", 10);
Dog d = new Dog("Fido", 4);

a = new Dog("Spot", 10);

d = a;
```

Compiler error because the static type of d is Dog and the static type of a is Animal. We can fix this by casting:

```
d = (Dog) a;
```

An Exercise in Inheritance Misery (Extra)

```
class C extends B {

    public int y = x + 1;

    public void m2() {System.out.println("Cm2-> " + super.x);}

     /*public void m4() {System.out.println("Cm4-> " +
    super.super.x); }} can't do super.super */

    public void m5() {System.out.println("Cm5-> " + y);}
 }
```

An Exercise in Inheritance Mystery (Extra)

\\ B a0 = new A();	Dynamic type must be B or subclass of B
\\ a0.m1();	cascading: prev line failed, so a0 can't be initialized
\\ a0.m2(16);	cascading: prev line failed, so a0 can't be initialized

An Exercise in Inheritance Mystery (Extra)

```
A b0 = new B();

System.out.println(b0.x);          [prints "5"]

b0.m1();                           [prints "Am1-> 5"]

b0.m2();                           [prints "Bm2-> 5"]

\\ b0.m2(61);                      m2 (int y) not defined in static type of b0
```

An Exercise in Inheritance Mystery (Extra)

```
B b1 = new B();

b1.m2(61);          [prints "Bm2y-> 61"]

b1.m3();            [prints "Bm3-> called"]

A c0 = new C();

c0.m2();            [prints "cm2-> 5"]

\\ C c1 = (A) new C(); Can't assign c1 to an A
```

An Exercise in Inheritance Mystery (Extra)

```
A c0 = new C();

A a1 = (A) c0;

C c2 = (C) a1;

c2.m3();                [print Bm3-> called]

\\ c2.m4();             C.m4() is invalid

c2.m5();                [print Cm5-> 6]
```

An Exercise in Inheritance Mystery (Extra)

```
A c0 = new C();

A b0 = new B();

((C) c0).m3();          [print Bm3-> called]

\\ (C) c0.m3();        NOT RUNTIME ERROR This would cast the result
of what the method returns and it returns void therefore
compile-time error

b0.update();

b0.m1();                [print Am1-> 99]
```