

## 1 Creating Cats

Given the `Animal` class, fill in the definition of the `Cat` class so that when `greet()` is called, the label "Cat" (instead of "Animal") is printed to the screen. Assume that a `Cat` will make a "Meow!" noise, and that this is all caps for cats younger than 5 years old.

```
1 public class Animal {
2     protected String name, noise;
3     protected int age;
4
5     public Animal(String name, int age) {
6         this.name = name;
7         this.age = age;
8         this.noise = "Huh?";
9     }
10
11    public String makeNoise() {
12        if (age < 5) {
13            return noise.toUpperCase();
14        } else {
15            return noise;
16        }
17    }
18
19    public void greet() {
20        System.out.println("Animal " + name + " says: " + makeNoise());
21    }
22 }

```

  

```
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());
    }
}

```

## 2 Raining Cats and Dogs

---

Assume that `Animal` and `Cat` are defined as above. What will be printed at each of the indicated lines?

```
1 public class TestAnimals {
2     public static void main(String[] args) {
3         Animal a = new Animal("Pluto", 10);
4         Cat c = new Cat("Garfield", 6);
5         Dog d = new Dog("Fido", 4);
6
7         a.greet();           // (A) _____
8         c.greet();           // (B) _____
9         d.greet();           // (C) _____
10
11        a = c;
12        ((Cat) a).greet();    // (D) _____
13        a.greet();           // (E) _____
14    }
15 }
16
17 public class Dog extends Animal {
18     public Dog(String name, int age) {
19         super(name, age);
20         noise = "Woof!";
21     }
22
23     @Override
24     public void greet() {
25         System.out.println("Dog " + name + " says: " + makeNoise());
26     }
27
28     public void playFetch() {
29         System.out.println("Fetch, " + name + "!");
30     }
31 }
```

- (A) Animal Pluto says: Huh?
- (B) Cat Garfield says: Meow!
- (C) Dog Fido says: WOOF!
- (D) Cat Garfield says: Meow!
- (E) Cat Garfield says: Meow!

Consider what would happen if we added the following to the bottom of `main`:

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

Why would this code produce a compiler error? How could we fix this error?

This code produces a compiler error in the second line. The `static` type of `d` is `Dog` while the `static` type of `a` is `Animal`. `Dog` is a subclass of `Animal`, so `this` assignment will fail at compile time because not all `Animals` are `Dogs`.

We can fix that by using a cast:

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

This represents a promise to the compiler that at runtime, `a` will be bound to an object that is compatible with the `Dog` type.

### 3 An Exercise in Inheritance Misery

Cross out any lines that cause compile-time errors, and put an X through runtime errors (if any). What does the main program (in class D) output after removing these lines? Note: There are many cases covered here and possibly not enough time to finish in discussion. Remember that solutions will be posted online later this week.

```
1 class A {
2     public int x = 5;
3     public void m1() {System.out.println("Am1-> " + x);}
4     public void m2() {System.out.println("Am2-> " + this.x);}
5     public void update() {x = 99;}
6 }
7
8 class B extends A {
9     public void m2() {System.out.println("Bm2-> " + x);}
10    public void m2(int y) {System.out.println("Bm2y-> " + y);}
11    public void m3() {System.out.println("Bm3-> ") + "called";}
12 }
13 class C extends B {
14     public int y = x + 1;
15     public void m2() {System.out.println("Cm2-> " + super.x);}
16     //public void m4() {System.out.println("Cm3-> " + super.super.x);} can't
17     //do super.super
18     public void m5() {System.out.println("Cm5-> " + y);}
19 }
20 class D {
21     public static void main (String[] args) {
22         <C> // B a0 = new A(); a0 must be B or a subclass of B.
23         <C> // a0.m1(); a0 is invalid
24         <C> // a0.m2(16); a0 is invalid
25         A b0 = new B();
26         System.out.println(b0.x); [5]
27         b0.m1(); [Am1-> 5]
28         b0.m2(); [Bm2-> 5]
29         <C> // b0.m2(61); m2(int y) not defined in static type
30         B b1 = new B();
31         b1.m2(61); [Bm2y-> 61]
32         b1.m3(); [Bm3-> called]
33         A c0 = new C();
34         c0.m2(); [cm2-> 5]
35         <C> // C c1 = (A) new C(); Can't assign c1 to an A.
36         A a1 = (A) c0;
37         C c2 = (C) a1;
38         c2.m3(); [Bm3-> called]
39         <C> // c2.m4(); C.m4() is invalid
40         c2.m5(); [Cm5-> 6]
41         ((C) c0).m3(); [Bm3-> called]
42         <C NOT R> // (C) c0.m3(); This would cast the result of what the
43         //method returns.
44         b0.update();
45         b0.m1(); [Am1-> 99]
46     }
47 }
```