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