# Classy Cats

Look at the `Animal` class defined below.

```java
public class Animal {
    protected String name, noise;
    protected int age;

    public Animal(String name, int age) {
        this.name = name;
        this.age = age;
        this.noise = "Huh?";
    }

    public String makeNoise() {
        if (age < 2) {
            return noise.toUpperCase();
        }
        return noise;
    }

    public String greet() {
        return name + ": " + makeNoise();
    }
}
```

(a) Given the `Animal` class, fill in the definition of the `Cat` class so that it makes a "Meow!" noise when `greet()` is called. Assume this noise is all caps for kittens, i.e. `Cats` that are less than 2 years old.

```java
public class Cat extends Animal {
    // Your implementation here
}
```
"Animal" is an extremely broad classification, so it doesn’t really make sense to have it be a class. Look at the new definition of the Animal class below.

```java
public abstract class Animal {
    protected String name;
    protected String noise = "Huh?";
    protected int age;

    public String makeNoise() {
        if (age < 2) {
            return noise.toUpperCase();
        }
        return noise;
    }

    public String greet() {
        return name + ": " + makeNoise();
    }

    public abstract void shout();
    abstract void count(int x);
}
```

Fill out the Cat class again below to allow it to be compatible with Animal (which is now an abstract class) and its two new methods.

```java
public class Cat extends Animal {
    public Cat() {
        this.name = "Kitty";
        this.age = 1;
        this.noise = "Meow!";
    }

    public Cat(String name, int age) {
        this();
        this.name = name;
        this.age = age;
    }

    @Override
    public String shout() {
        System.out.println(noise.toUpperCase());
    }

    @Override
    public abstract void count(int x) {
        for (int i = 0; i < x; i++) {
            System.out.println(makeNoise());
        }
    }
}
```
2 The Interfacing CatBus

After discovering that we can implement the Cat class with minimal effort, Professor Hilfinger decided that he wants to create a CatBus class. CatBuses are Cats that act like vehicles and have the ability to honk (safety is important!).

a) Given the Vehicle and Honker interfaces, fill out the CatBus class so that CatBuses can rev their engines and honk at other CatBuses.

```java
interface Vehicle {
    /** Gotta go fast! */
    public void revEngine();
}

interface Honker {
    /** HONQUE! */
    void honk();
}

public class CatBus extends _______, implements __________, __________ {

    @Override
    __________ __________ revEngine() {
        System.out.println("Purrrrrrr");
    }

    @Override
    __________ __________ honk() {
        System.out.println("CatBus says HONK");
    }

    /** Allows CatBus to honk at other CatBuses. */
    public void conversation(CatBus target, int duration) {
        for (int i = 0; i < duration; i++) {
            honk();
            target.honk();
        }
    }
}
```

b) After a few hours of research, Professor Hilfinger discovered that animals of type Goose are also avid Honkers! Modify the conversation method so that CatBuses can honk at CatBuses and Geese.

```java
/** Allows CatBus to honk at ANY target that can honk back. */

public void conversation(___________ target, int duration) {
    for (int i = 0; i < duration; i++) {
        honk();
        target.honk();
    }
}
```
3 Raining Cats & Dogs

In addition to Animal and Cat from Problem 1a, we now have the Dog class! (Assume that the Cat and Dog classes are both in the same file as the Animal class.)

class Dog extends Animal {
    public Dog(String name, int age) {
        super(name, age);
        noise = "Woof!";
    }
    public void playFetch() {
        System.out.println("Fetch, " + name + "!");
    }
}

Consider the following main function in the Animal class. Decide whether each line causes a compile time error, a runtime error, or no error. If a line works correctly, draw a box-and-pointer diagram and/or note what the line prints. It may be useful to refer to the Animal class back on the first page.

public static void main(String[] args) {
    Cat nyan = new Animal("Nyan Cat", 5); (A) ________________
    Animal a = new Cat("Olivia Benson", 3); (B) ________________
    a = new Dog("Fido", 7); (C) ________________
    System.out.println(a.greet()); (D) ________________
    a.playFetch(); (E) ________________
    Dog d1 = a; (F) ________________
    Dog d2 = (Dog) a; (G) ________________
    d2.playFetch(); (H) ________________
    (Dog) a.playFetch(); (I) ________________
    Animal imposter = new Cat("Pedro", 12); (J) ________________
    Dog fakeDog = (Dog) imposter; (K) ________________
    Cat failImposter = new Cat("Jimmy", 21); (L) ________________
    Dog failDog = (Dog) failImposter; (M) ________________
}
4 Bonus: An Exercise in Inheritance Misery

Cross out any lines that cause compile or runtime errors. What does the main program output after removing those lines?

Moral of the story: Fields become hidden when you redefine them in the subclass. If possible, you should avoid doing so or else your code may become confusing.

```java
1 class A {
2    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 class B extends A {
8    int x = 10;
9    public void m2() {System.out.println("Bm2-> " + x);}  
10   public void m3() {System.out.println("Bm3-> " + super.x);}  
11   public void m4() {System.out.print("Bm4-> "); super.m2();}
12 }
13 class C extends B {
14    int y = x + 1;
15    public void m2() {System.out.println("Cm2-> " + super.x);}  
16    public void m3() {System.out.println("Cm3-> " + super.super.x);}  
17    public void m4() {System.out.println("Cm4-> " + y);}  
18    public void m5() {System.out.println("Cm5-> " + super.y);}
19 }
20 class D {
21   public static void main (String[] args) {
22      A b0 = new B();
23      System.out.println(b0.x);  (A) ________________
24      b0.m1();                   (B) ________________
25      b0.m2();                   (C) ________________
26      b0.m3();                   (D) ________________
27      B b1 = new B();
28      b1.m3();                   (E) ________________
29      b1.m4();                   (F) ________________
30      A c0 = new C();
31      c0.m1();                   (G) ________________
32      A a1 = (A) c0;           
33      C c2 = (C) a1;           
34      c2.m4();                 (H) ________________
35      ((C) c0).m3();           (I) ________________
36      b0.update();            
37      b0.m1();                 (J) ________________
38    }
39 } 
```