

1 Classy Cats

Look at the `Animal` class defined below.

```
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 < 2) {
13            return noise.toUpperCase();
14        }
15        return noise;
16    }
17
18    public String greet() {
19        return name + ": " + makeNoise();
20    }
21 }
```

- (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.

```
public class Cat extends Animal {

}
```

(b) "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.

```
1 public abstract class Animal {
2     protected String name;
3     protected String noise = "Huh?";
4     protected int age;
5
6     public String makeNoise() {
7         if (age < 2) {
8             return noise.toUpperCase();
9         }
10        return noise;
11    }
12
13    public String greet() {
14        return name + ": " + makeNoise();
15    }
16
17    public abstract void shout();
18    abstract void count(int x);
19 }
```

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.

```
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
    _____ shout() {
        System.out.println(noise.toUpperCase());
    }

    @Override
    _____ 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`.

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

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

public class CatBus extends _____, implements _____, _____ {

    @Override
    _____ _____ revEngine() {
        System.out.println("Purrrrrrrr");
    }

    @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`s.

```
/** 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.)

```
1 class Dog extends Animal {
2     public Dog(String name, int age) {
3         super(name, age);
4         noise = "Woof!";
5     }
6     public void playFetch() {
7         System.out.println("Fetch, " + name + "!");
8     }
9 }
```

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.

```
1 public static void main(String[] args) {
2     Cat nyan = new Animal("Nyan Cat", 5); (A) _____
3
4     Animal a = new Cat("Olivia Benson", 3); (B) _____
5     a = new Dog("Fido", 7); (C) _____
6     System.out.println(a.greet()); (D) _____
7     a.playFetch(); (E) _____
8
9     Dog d1 = a; (F) _____
10    Dog d2 = (Dog) a; (G) _____
11    d2.playFetch(); (H) _____
12    (Dog) a.playFetch(); (I) _____
13
14    Animal imposter = new Cat("Pedro", 12); (J) _____
15    Dog fakeDog = (Dog) imposter; (K) _____
16
17    Cat failImposter = new Cat("Jimmy", 21); (L) _____
18    Dog failDog = (Dog) failImposter; (M) _____
19 }
```

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.

```
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
28         B b1 = new B();
29         b1.m3();                       (E) _____
30         b1.m4();                       (F) _____
31
32         A c0 = new C();
33         c0.m1();                       (G) _____
34
35         A a1 = (A) c0;
36         C c2 = (C) a1;
37         c2.m4();                       (H) _____
38         ((C) c0).m3();                 (I) _____
39
40         b0.update();
41         b0.m1();                       (J) _____
42     }
43 }
```