

1 Creating Cats

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 (less than 2 years old).

```

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

```

```

class Cat extends Animal {
}

```

2 Impala-ments

a) We have two interfaces, `BigBaller` and `ShotCaller`. We also have `LilTroy`, a concrete class, which should implement `BigBaller` and `ShotCaller`. Fill out the blank lines below so that the code compiles correctly.

```

1 interface BigBaller {
2     void ball();
3 }
4 interface ShotCaller {
5     void callShots();
6 }
7 public class LilTroy _____, _____ {
8     public void ball() {
9         System.out.println("Wanna be a, baller");
10    }
11    public void callShots() {
12        System.out.println("Shot caller");
13    }
14    public void rap() {
15        System.out.println("Say: Twenty inch blades on the Impala");
16    }
17 }

```

b) We have a `BallCourt` where ballers should be able to come and play. However, the below code demonstrates an example of bad program design. Right now, only `LilTroy` instances can ball.

```
1 public class BallCourt {
2     public void play(LilTroy lilTroy) {
3         lilTroy.ball();
4     }
5 }
```

Fix the `play` method so that all the `BigBallers` can ball.

```
public class BallCourt {
    public void play(_____ _____) {
        _____
    }
}
```

c) We discover that Rappers have some common behaviors, leading to the following class.

```
1 class Rapper {
2     public abstract String getLine();
3     public final void rap() {
4         System.out.println("Say: " + getLine());
5     }
6 }
```

Will the above class compile? If not, why not? How can we fix it?

d) Rewrite `LilTroy` so that `LilTroy` extends `Rapper` and displays exactly the same behavior as in part a) *without* overriding the `rap` method (in fact, you *cannot* override final methods).

```
public class LilTroy extends _____ implements _____, _____ {

}
```

3 Raining Cats & Dogs

In addition to `Animal` and `Cat` from Problem 1, 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.

```
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 are hidden if also defined in the subclass, and therefore you should avoid doing that because it makes the code 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 }
```