1  Raining Cats and Dogs

Write out the necessary classes for the code below to compile and produce the given output.

OUTPUT:
Dug wolfs down the food. Woof!
Pluto sniffs the food. Woof!
Salem sniffs the food. Meow.
Nyan Cat wolfs down the food. Meow.

class TestAnimals {
    public static void main(String[] args) {
        Dog dug = new Dog("Dug", 5);
        Dog pluto = new Dog("Pluto", 84);
        Cat salem = new Cat("Salem", 18);
        Cat nyancat = new Cat("Nyan Cat", 3);
        Animal[] animals = {dug, pluto, salem, nyancat};
        feed(animals);
    }

    public static void feed(Animal[] animals) {
        for (Animal a : animals) {
            if (a.getAge() < 10) {
                System.out.print(a.getName() + " wolfs down the food. ");
            } else {
                System.out.print(a.getName() + " sniffs the food. ");
            }
            a.makeSound();
        }
    }
}

public abstract class Animal {
    private String _name;
    private int _age;
    public Animal(String name, int age) {
        _name = name;
        _age = age;
    }

    public int getAge() { return _age; }
    public String getName() { return _name; }
    public abstract void makeSound();
}

public class Dog extends Animal {
    public Dog(String name, int age) { super(name, age); }
    //mention (at)Override tag
    public void makeSound() { System.out.println("Woof!"); }
}

public class Cat extends Animal {
    public Cat(String name, int age) { super(name, age); }
    public void makeSound() { System.out.println("Meow."); }
}
Discussion Question: Should Animal be an interface or an abstract class?

- Animal should be an abstract class if there are methods that all animals should get and are implemented the same way like getName() and getAge().

- Look how short the Dog and Cat class are above! If this were an interface, we would have to reimplement getAge() and getName().

- A downside to Animal as an abstract class is that Dog and Cat might want to inherit from a different abstract class. You can only inherit from one abstract class, but you can implement multiple interfaces.

Output of Question 2 (Next Page):

Am1-> 5
Bm2-> 10
Bm3-> 5
Bm4-> Am2-> 5
Am1-> 5
Cm4-> 11
Bm3-> 5
Am1-> 99
Bm2-> 10

Individual outputs are next to their corresponding line and in brackets.
Cross out any lines that cause compile-time or runtime errors. What does the main program (in class D) output after removing these lines?

```java
class A {
    int x = 5;
    public void m1() {System.out.println("Am1-> " + x);}  
    public void m2() {System.out.println("Am2-> " + this.x);}  
    public void update() {x = 99;}
}

class B extends A {
    int x = 10;
    public void m2() {System.out.println("Bm2-> " + x);}  
    public void m3() {System.out.println("Bm3-> " + super.x);}  
    public void m4() {System.out.println("Bm4-> "); super.m2();}
}

class C extends B {
    int y = x + 1;
    public void m2() {System.out.println("Cm2-> " + super.x);}  
    //public void m3() {System.out.println("Cm3-> " + super.super.x);}  
    ^ can’t do super.super  
    public void m4() {System.out.println("Cm4-> " + y);}  
    //public void m5() {System.out.println("Cm5-> " + super.y); B class has no y
}

class D {
    public static void main (String[] args) {
        //B a0 = new A(); a0 must be B or a subclass of B.
        //a0.m1(); a0 is invalid
        A b0 = new B();
        b0.m1(); [Am1-> 5]
        b0.m2(); [Bm2-> 10]
        //b0.m3(); Can only call a method which the static type has.
        B b1 = new B();
        b1.m3(); [Bm3-> 5]
        b1.m4(); [Bm4-> Am2-> 5]
        A c0 = new C();
        c0.m1(); [Am1-> 5]
        //C c1 = (A) new C(); Correct casting syntax. Can’t assign c1 to an A.
        A a1 = (A) c0;
        C c2 = (C) a1;
        c2.m4(); [Cm4-> 11]
        //c2.m5(); m5 is invalid
        ((C) c0).m3(); [Bm3-> 5]
        //((C) c0.m3()); This would cast the result of what the method returns.
        b0.update();
        b0.m1(); [Am1-> 99]
        b0.m2(); [Bm2-> 10]
    }
}
```

CS 61B, Fall 2014, Discussion 4: Inheritance