CS 61B Spring 2018

Inheritance

Discussion 4: February 6, 2018

1 Creating Cats

1.1 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 if the cat is 5 years or older and "MEOW!" if the cat is less than 5 years old.

```
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 < 5) {
            return noise.toUpperCase();
        } else {
            return noise;
        }
    }

    public void greet() {
        System.out.println("Animal " + name + " says: " + makeNoise());
    }
}
```

```
    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

2.1 Assume that `Animal` and `Cat` are defined as above. What would Java print on 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          a.greet();          // (A) Animal Pluto says: Huh?
7          c.greet();          // (B) Cat Garfield says: Meow!
8          d.greet();          // (C) Dog Fido says: WOOF!
9          a = c;
10         ((Cat) a).greet();  // (D) Cat Garfield says: Meow!
11         a.greet();          // (E) Cat Garfield says: Meow!
12     }
13 }
14 public class Dog extends Animal {
15     public Dog(String name, int age) {
16         super(name, age);
17         noise = "Woof!";
18     }
19     @Override
20     public void greet() {
21         System.out.println("Dog " + name + " says: " + makeNoise());
22     }
23     public void playFetch() {
24         System.out.println("Fetch, " + name + "!");
25     }
26 }
```

2.2 Consider what would happen if we added the following to the bottom of `main` under line 12:

```
1  a = new Dog("Spot", 10);
2  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`. Use casting to address the problem.

```
1  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 *Extra*

3.1 Cross out any lines that cause compile-time errors or cascading errors (failures that occur because of an error that happened earlier in the program), and put an X through runtime errors (if any). Don't just limit your search to main, there could be errors in classes A,B,C. What does `D.main` output after removing these lines?

```
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  class B extends A {
8      public void m2() {System.out.println("Bm2-> " + x);}
9      public void m2(int y) {System.out.println("Bm2y-> " + y);}
10     public void m3() {System.out.println("Bm3-> " + "called");}
11 }
12 class C extends B {
13     public int y = x + 1;
14     public void m2() {System.out.println("Cm2-> " + super.x);}
15     \\ public void m4() {System.out.println("Cm4-> " + super.super.x); }} can't do super.super
16     public void m5() {System.out.println("Cm5-> " + y);}
17 }
18 class D {
19     public static void main (String[] args) {
20         \\ B a0 = new A(); Dynamic type must be B or subclass of B
21         \\ a0.m1(); cascading: prev line failed, so a0 can't be initialized
22         \\ a0.m2(16); cascading: prev line failed, so a0 can't be initialized
23         A b0 = new B();
24         System.out.println(b0.x); [prints "5"]
25         b0.m1(); [prints "Am1-> 5"]
26         b0.m2(); [prints "Bm2-> 5"]
27         \\ b0.m2(61); m2 (int y) not defined in static type of b0
28         B b1 = new B();
29         b1.m2(61); [prints "Bm2y-> 61"]
30         b1.m3(); [prints "Bm3-> called"]
31         A c0 = new C();
32         c0.m2(); [prints "cm2-> 5"]
33         \\ C c1 = (A) new C(); Can't assign c1 to an A
34         A a1 = (A) c0;
35         C c2 = (C) a1;
36         c2.m3(); [print Bm3-> called]
37         \\ c2.m4(); C.m4() is invalid
38         c2.m5(); [print Cm5-> 6]
39         ((C) c0).m3(); [print Bm3-> called]
```

```
40          \\ (C) c0.m3(); NOT RUNTIME ERROR This would case the result of what the method returns and
        it returns void therefore compile-time error
41          b0.update();
42          b0.m1(); [print Am1-> 99]
43      }
44  }
```