CS 61B Discussion 4: Inheritance Fall 2016

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 age) {
5          this.name = name;
6          this.age = age;
7          this.noise = "Huh?";
8      }
9      public String makeNoise() {
10         if (age < 2) {
11             return noise.toUpperCase();
12         }
13         return noise;
14     }
15     public String greet() {
16         return name + ": " + makeNoise();
17     }
18 }

class Cat extends Animal {









}
```

2 Impala-ments

a) We have two interfaces, `BigBaller` and `ShotCaller`. `LilTroy`, a concrete class, 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");
```

```
    }
    public void rap() {
        System.out.println("Say: Twenty inch blades on the Impala");
    }
}
```

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` can ball.

```
public class BallCourt {
    public void play(LilTroy lilTroy) {
        lilTroy.ball();
    }
}
```

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.

```
class Rapper {
    public abstract String getLine();
    public final void rap() {
        System.out.println("Say: " + getLine());
    }
}
```

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).

```
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.

```
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?

```
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 }
```