CS 61B Discussion 4: Inheritance Fall 2018

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).

```java
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 < 2) {
            return
                noise.toUpperCase();
        }
        return noise;
    }
    public String greet() {
        return name + ": " +
            makeNoise();
    }
}
```

```java
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.

```java
interface BigBaller {
    void ball();
}
interface ShotCaller {
    void callShots();
}
public class LilTroy ______________  ______________,______________ {
    public void ball() {
        System.out.println("Wanna be a, baller");
    }
    public void callShots() {
        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` instances 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).

```
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.)

```java
class Dog extends Animal {
    public Dog(String name, int age) {
        super(name, age);
        noise = "Woof!";
    }
    public void playFetch() {
        System.out.println("Fetch, " + name + "!");
    }
}
```

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.

```java
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?

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.

```
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.print("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);}
    public void m4() {System.out.println("Cm4-> " + y);}
    public void m5() {System.out.println("Cm5-> " + super.y);}
}
class D {
    public static void main (String[] args) {
        A b0 = new B();
        System.out.println(b0.x);    (A) ____________________
        b0.m1();                     (B) ____________________
        b0.m2();                     (C) ____________________
        b0.m3();                     (D) ____________________

        B b1 = new B();
        b1.m3();                     (E) ____________________
        b1.m4();                     (F) ____________________

        A c0 = new C();
        c0.m1();                     (G) ____________________

        A a1 = (A) c0;
        C c2 = (C) a1;
        c2.m4();                     (H) ____________________
        ((C) c0).m3();               (I) ____________________

        b0.update();
        b0.m1();                     (J) ____________________
    }
}
```