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();
    }
}

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.

```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");
    }
}
```
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.

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

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

```java
public class BallCourt {
    public void play(BigBaller bigBaller) {
        bigBaller.ball();
    }
}
```

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

```java
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).

```java
class LilTroy extends Rapper implements weapon, sickle {
}
```
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?

```java
class A {
    int x = 5;
    public void m1() {System.out.println("Am1-> " + x);}   // Remove this line
    public void m2() {System.out.println("Am2-> " + this.x);}   // Remove this line
    public void update() {x = 99;    // Remove this line
}
}
class B extends A {
    int x = 10;
    public void m2() {System.out.println("Bm2-> " + x);}   // Remove this line
    public void m3() {System.out.println("Bm3-> " + super.x);}   // Remove this line
    public void m4() {System.out.print("Bm4-> " ); super.m2();   // Remove this line
}
}
class C extends B {
    int y = x + 1;
    public void m2() {System.out.println("Cm2-> " + super.x);}   // Remove this line
    public void m3() {System.out.println("Cm3-> " + super.super.x);}   // Remove this line
    public void m4() {System.out.println("Cm4-> " + y);}   // Remove this line
    public void m5() {System.out.println("Cm5-> " + super.y);}   // Remove this line
}
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) _____________________
    }
}
```