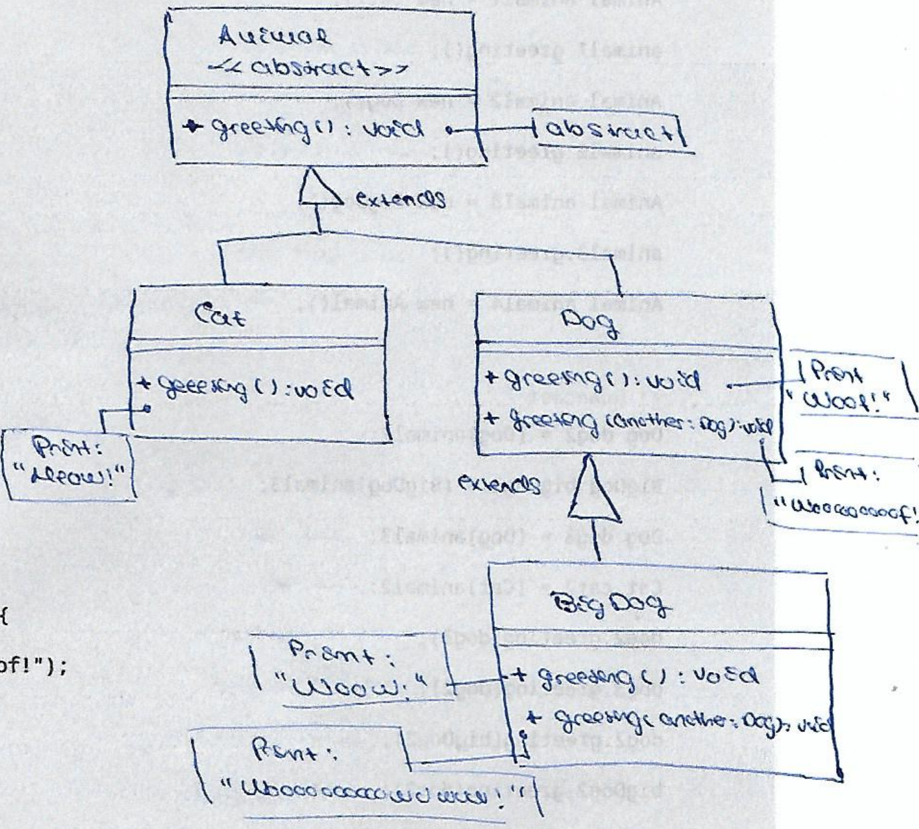
6.7 Implementación alternativa

A partir del siguiente código, genere el diagrama de clases UML correspondiente.

```
abstract public class Animal {
    abstract public void greeting();
}
public class Cat extends Animal {
    @Override
    public void greeting() {
        System.out.println("Meow!");
    }
}
public class Dog extends Animal {
    @Override
    public void greeting() {
        System.out.println("Woof!");
    }


    public void greeting(Dog another) {
        System.out.println("Woooooooooof!");
    }
}
public class BigDog extends Dog {
    @Override
    public void greeting() {'
        System.out.println("Woow!");
    }


    @Override
    public void greeting(Dog another) {
        System.out.println("Wooooowwwww!");
    }
}
```



Explique las salidas que se obtendrían con el siguiente código:

```
public class TestAnimal {
    public static void main(String[] args) {
        // Using the subclasses
        Cat cat1 = new Cat();
        cat1.greeting(); → // Meow!
        Dog dog1 = new Dog();
        dog1.greeting(); → // Woof!
        BigDog bigDog1 = new BigDog();
        bigDog1.greeting(); → // Woow!

        // Using Polymorphism
        Animal animal1 = new Cat();
        animal1.greeting(); → // Meow!
        Animal animal2 = new Dog();
        animal2.greeting(); → // Woof!
        Animal animal3 = new BigDog();
        animal3.greeting(); → // Woow!
        Animal animal4 = new Animal(); → // ERROR. (Animal es abstracta ⇒ no se puede
                                            usar directamente).

        // Downcast
        Dog dog2 = (Dog)animal2;
        BigDog bigDog2 = (BigDog)animal3; → *
        Dog dog3 = (Dog)animal3; → *
        Cat cat2 = (Cat)animal2; → *
        dog2.greeting(dog3); → // Excepción
        dog3.greeting(dog2); → // Excepción
        dog2.greeting(bigDog2); → // Excepción
        bigDog2.greeting(dog2); → // Excepción
        bigDog2.greeting(bigDog1); → // Excepción
    }
}
```