The Interfaces in our Application

```java
public interface Speakable {
    public String speak();
}

public interface Movable {
    public String move();
}

public interface Animal extends Movable, Speakable {
    public String getName();
}
```

The Classes in our Application

```java
public class Dog implements Animal {
    private String name;
    
    public Dog(String itsName) {
        name = itsName;
    }
    
    public String speak() {
        return "Bark!";
    }
    
    public String move() {
        return "Run, run."
    }
    
    public String getName() {
        return name;
    }
}

public class Cat implements Animal {
    private String name;
    
    public Cat(String itsName) {
        name = itsName;
    }
    
    public String speak() {
        return "Meow!
    }
    
    public String move() {
        return "Walk, walk."
    }
    
    public String getName() {
        return name;
    }
}
```

The Main Program in our System

```java
public class AnimalApp {
    public static void main(String[] args) {
        ArrayList<Animal> myPets = new ArrayList<Animal>();
        myPets.add(new Dog("Life Saver");
        myPets.add(new Cat("Annoying");
        
        for (Animal theAnimal : myPets) {
            System.out.println("My name is " + theAnimal.getName() + ". ");
            System.out.println("When I speak I say, " + theAnimal.speak() + ", and ");
            System.out.println("when I move I, " + theAnimal.move() + 
        }
    }
}
```

Using the Adapter Pattern

Suppose we find an API for a snake class. These are the methods listed in the API:

```java
class Snake {
    public String slither();
    public String hiss();
    public String getName();
}
```

We want to add snake objects to our program, and we want to reuse the snake class. Develop a solution using the adapter pattern.