There are animals in the zoo: Cat, dog, elephant.
Each animal has a color and a name.
All animals have four legs.
Cats say meow, dogs say woof, elephants say woo.
I want to be able to

- add new animals (of existing or new types) to the zoo.
- make all the animals make sound.
- count the total number of legs.

Version 0:

```
public class Cat {
    private int color;
    private String name;

    public Cat(String name, int color) {
        this.name = name;
        this.color = color;
    }

    public int color() {
        return color;
    }

    public String name() {
        return name;
    }

    public int numLegs() {
        return 4;
    }

    public public String sound() {
        return "meow";
    }
}

public class Dog {
    private int color;
    private String name;

    public Dog(String name, int color) {
        this.name = name;
        this.color = color;
    }

    public int color() {
        return color;
    }

    public String name() {
        return name;
    }

    public int numLegs() {
        return 4;
    }

    public String sound() {
        return "woof";
    }
}

public class Elephant {
    private int color;
    private String name;

    public Elephant(String name, int color) {
        this.name = name;
        this.color = color;
    }
```

```
    public int color() {
        return color;
    }

    public String name() {
        return name;
    }

    public int numLegs() {
        return 4;
    }

    public String sound() {
        return "woo";
    }
}

public class Zoo {
    private List animals = new ArrayList();

    public void add(Object animal) {
        animals.add(animal);
    }

    public void allSounds() {
        for (Object animal: animals) {
            String sound;
            if (animal instanceof Cat) {
                sound = ((Cat)animal).sound();
            } else if (animal instanceof Dog) {
                sound = ((Dog)animal).sound();
            } else {
                sound = ((Elephant)animal).sound();
            }
            System.out.println(sound);
        }
    }

    public void allLegs() {
        int sum = 0;
        for (Object animal: animals) {
            int legs = 0;
            if (animal instanceof Cat) {
                legs = ((Cat)animal).numLegs();
            } else if (animal instanceof Dog) {
                legs = ((Dog)animal).numLegs();
            } else {
                legs = ((Elephant)animal).numLegs();
            }
            sum += legs;
        }
        System.out.println("Counted " + sum + " legs.");
    }
}

public class Main {
    public static void main(String[] args) {
        Zoo zoo = new Zoo();
        zoo.add(new Cat("Felix", 123));
        zoo.add(new Dog("Karabas", 99));
        zoo.add(new Dog("Rocky", 256));
        zoo.add(new Elephant("Dumbo", 3));
        zoo.add(new Cat("Pamuk", 456));
        zoo.add(new Elephant("Elmar", 5));

        zoo.allSounds();
        zoo.allLegs();
    }
}
```

There are major problems in this version regarding code duplication and maintainability. Try to add a Chicken class and see how much code you need to add/alter. Also, we have to do lots of type-casting; there is nothing that prevents us from adding a Carrot or a GradStudent into the Zoo.

Version 1:

```
public interface Animal {
    public int color();

    public String name();

    public int numLegs();

    public String sound();
}

public class Cat implements Animal {
    … same as Version 0
}

public class Dog implements Animal {
    … same as Version 0
}

public class Elephant implements Animal {
    … same as Version 0
}

public class Zoo {
    private List<Animal> animals = new ArrayList<Animal>();

    public void add(Animal animal) {
        animals.add(animal);
    }

    public void allSounds() {
        // We see a use of polymorphism below
        // Each animal is accessed through the
        // Animal interface, although the objects
        // are of different types, such as Cat and Dog.
        for (Animal animal: animals) {
            String sound = animal.sound();
            System.out.println(sound);
        }
    }

    public void allLegs() {
        int sum = 0;
        for (Animal animal: animals) {
            sum += animal.numLegs();
        }
        System.out.println("Counted " + sum + " legs.");
    }
}
public class Main {
   … same as Version 0. Zoo interface is the same, so the client code did not break.
}
```

Now, the Zoo class has been improved greatly.

Version 2:

```
public abstract class Animal {
    private int color;
    private String name;

    public Animal(String name, int color) {
        this.name = name;
        this.color = color;
    }

    public int color() {
        return color;
    }

    public String name() {
        return name;
    }

    public int numLegs() {
        return 4;
    }

    public abstract String sound();
}

// THINK: Why not this?
/*
abstract class Animal {
    private int color;
    private String name;
    private int numLegs;

    abstract String sound();

    int color() {
        return color;
    }

    String name() {
        return name;
    }

    int numLegs() {
        return numLegs;
    }
}
*/

public class Cat extends Animal {
    public Cat(String name, int color) {
        super(name, color);
    }

    public String sound() {
        return "meow";
    }
}

public class Dog extends Animal {
    public Dog(String name, int color) {
        super(name, color);
    }

    public String sound() {
        return "woof";
    }
}

public class Elephant extends Animal {
    public Elephant(String name, int color) {
        super(name, color);
    }
```

```
    public String sound() {
        return "woo";
    }
}
```

This version solves the code duplication problem. Also, extension with a new animal type is straightforward. Simply add a new class, you don't need to touch Animal, Cat, Dog, Elephant, Zoo classes.

```
public class Dog extends Animal {
    public Dog(String name, int color) {
        super(name, color);
    }

    public String sound() {
        return "woof";
    }

    public int numLegs() {
        return 2;
    }
}
```

THINK: What if I wanted to add a new animal action, e.g. a walk() method?
Would this change be isolated or scattered over multiple locations?

Version 3:
Let's see what would happen if we had a radically different design. Delete the Cat, Dog, Elephant classes. There is only the Animal class:

```
enum Type {
    CAT, DOG, ELEPHANT
}

public class Animal {
    private Type type;
    private int color;
    private String name;

    public Animal(Type type, String name, int color) {
        this.type = type;
        this.name = name;
        this.color = color;
    }

    public int color() {
        return color;
    }

    public String name() {
        return name;
    }

    public int numLegs() {
        return 4;
    }

    public String sound() {
        switch(type) {
            case CAT: return "meow";
            case DOG: return "woof";
            case ELEPHANT: return "woo";
            default: throw new Error();
        }
    }
}

public class Zoo {
   … same as Version 2.
}

public class Main {
    public static void main(String[] args) {
        Zoo zoo = new Zoo();
        zoo.add(new Animal(Type.CAT, "Felix", 123));
        zoo.add(new Animal(Type.DOG, "Karabas", 99));
        zoo.add(new Animal(Type.DOG, "Rocky", 256));
        zoo.add(new Animal(Type.ELEPHANT, "Dumbo", 3));
        zoo.add(new Animal(Type.CAT, "Pamuk", 456));
        zoo.add(new Animal(Type.ELEPHANT, "Elmar", 5));

        zoo.allSounds();
        zoo.allLegs();
    }
}
```

Version 4:
Suppose we want to add the walk() method. How do we do that? Simply add the following method to the Animal class. This is a modular extension, the operation's definition is all in one place; it's not scattered.

```
public void run() {
    switch(type) {
        case CAT:
            System.out.println("cat walks..."); break;
        case DOG:
            System.out.println("dog walks..."); break;
        case ELEPHANT:
            System.out.println("elephant walks..."); break;
        default: throw new Error();
    }
}
```

HOWEVER, how does this version handle extending the system with a new animal type? The modification is not modular. Chicken behavior is scattered over multiple places.

```
enum Type {
    CAT, DOG, ELEPHANT, CHICKEN
}

public class Animal {
    private Type type;
    private int color;
    private String name;

    public Animal(Type type, String name, int color) {
        this.type = type;
        this.name = name;
        this.color = color;
    }

    public int color() {
        return color;
    }

    public String name() {
        return name;
    }

    public int numLegs() {
        if (type == Type.CHICKEN)
            return 2;
        return 4;
    }

    public String sound() {
        switch(type) {
            case CAT: return "meow";
            case DOG: return "woof";
            case ELEPHANT: return "woo";
            case CHICKEN: return "gidak gidak";
            default: throw new Error();
        }
    }

    public void run() {
        switch(type) {
            case CAT:
                System.out.println("cat walks..."); break;
            case DOG:
                System.out.println("dog walks..."); break;
            case ELEPHANT:
                System.out.println("elephant walks..."); break;
            case CHICKEN:
                System.out.println("elephant walks..."); break;
            default: throw new Error();
        }
    }
}
```