某一天，项目经理阿呆兴冲冲的说，我们要做一款惊爆的手游，名字暂且叫《变形虫》吧。

然后，叫着我们几个程序员，一起看他的原型图。

> 一个初代变形虫，有名字，有身长，有重量，虫龄

这几个程序员里，阿狗之前是做C语言的， 阿猫之前是做JAVA的。项目经理就说，阿猫阿狗你们各自做吧，到时候我看看谁的好来采用。

于是，两个有志向的程序开始大干了。

阿猫的行动

阿猫：C语言是面向过程的，很简单啊。先定义一个变形虫的结构体

```
struct worm{
    char *name;
    float weight;
    float length;
    int age;
};
```

有了结构体，我就可以生产初代变形虫了

```
void main(){
    struct worm worm1;

    worm1.name= "alis";
    worm1.age="1"
    worm1.length="0.5"
    worm1.weight="1"

    printf("初代变形虫，名字%s，虫龄%d，长%d，重%d", worm1.name, worm1.age, worm1.length, worm1.weight);
}
```

大功告成。

阿狗的行动

java是典型的面向对象语言，所以，我先定义一个类，名字叫worm

```
public class Worm{
   public String name;
   public float weight;
   public float length;
   public int age;

   public Worm(String name, float weight, float length, int age){
      this.name = name;
      this.weight = weight;
      this.length = length;
      this.age = age;
   }
}
```

类定义好了，我直接调用就好了

```
Worm worm1= new Worm("tom", 1.1, 2.0, 1);
System.out.println("初代变形虫:名字"+worm1.name+",长"+worm1.length+",重"+"worm1.weight"+",虫龄"+worm1.age);
```

第一版开发，阿猫阿狗的代码在简洁度和实现上看似打了个平手。

然而，**可怕的需求更改来了，阿呆说：**

> **这个初代变形虫要发生变异了，就叫一代变形虫吧，一代变形虫会发出可怕叫声，会呼啦的爬**

阿狗一看，很简单哟，我在再定义一个一代变形虫里加个叫声和爬的定义就好

阿狗的行动

```
struct worm_first{
    char *name, *sound;
```

```
    float weight, speed;
    float length;
    int age;

};
```

一代变形虫调用我多加两个参数进来就好了

```
void main(){
    struct worm_first worm1;

    worm1.name= "alis";
    worm1.age="1";
    worm1.length="0.5";
    worm1.weight="1";
    worm1.sound ="wawawa";
    worm1.speed = 1;

    printf("一代变形虫，名字%s，虫龄%d，长%d，重%d，叫声%s，爬速%d",
worm1.name, worm1.age, worm1.length, worm1.weight,  worm1.sound, worm1.speed);
}
```

阿狗自信的完成了

阿猫的行动

面向对象第一大法：**继承**

看我展示，我只需要继承初代变形虫，加两个定义，不用像阿狗那样重新定义了，绝对简单明了。于是阿猫写下来这个。

```
public class Worm_first extends Worm{
    public String sound;
    public int speed;
    public Worm_first(String name, float weight, float length, int age, String sound, int speed){
        super(name, weight, length, age);
        this.sound = sound;
        this.speed = speed;
```

```
    }
}
```

调用直接

```
Worm_first worm1= new Worm_first("tom", 1.1, 2.0, 1, 'huluhulu', 1);
System.out.println("初代变形虫:名字"+worm1.name+",长"+worm1.length+",重"+"worm1.weight"+",虫龄"+worm1.age+
",叫声"+worm1.sound+",爬速"+worm1.speed);
```

这一回合，貌似阿猫略胜一筹。不过作为一个产品经理，阿呆当然还有很多奇怪的想法。

阿呆这时候又说了

> **一代变形虫的寿命不能超过五年，要是超过五年他会重生**

阿狗看着这个需求，挠了挠头。

看来我得定义一个函数来判断赋值一代变形虫的年龄了，于是阿狗写下了这个。

阿狗的行动

```
int replace_age(int age);
int replace_age(int age)
{
    if(age > 5){
      return 0;
    }else{
       return age;
    }
}
void main(){
    struct worm_first worm1;

    worm1.name= "alis";
    worm1.age= replace_age(4);
    worm1.length="0.5";
```

```
        worm1.weight="1";
        worm1.sound ="wawawa";
        worm1.speed = 1;

        printf("               %s,     %d,   %d,   %d,     %s      %d",
    worm1.name, worm1.age, worm1.length, worm1.weight   worm1.sound, worm1.speed);
    }
```

虽然有点凌乱，但是看似还行

阿猫看了看，眼睛一转，这不是就是面向对象的 **封装** 大法吗, 我把年龄设为私有变量，这样就不能随意改变了，只有调用set_age方法才能去设置年龄。

```
public class Worm_first extends Worm{
    public String sound;
    public int speed;
    private int age;

    public void set_age(int age){
        if(age > 5){
           this.age = 0;
        }else{
           this.age = age;
        }
    }

    public Worm_first(String name, float weight, float length,  String sound, int speed){
        super(name, weight, length);
        this.sound = sound;
        this.speed = speed;
    }

}
```

项目到这，大家都开始放假，先休息一段时间。

这时候新来了一个程序员铁蛋，开始维护这两份代码。这时候。

封装的伟大之处出现了

由于不知道之前的需求，铁蛋有一天做了测试，给一代变形虫赋值了10岁的年龄

对于阿狗的代码,铁蛋没有看到那个判断年龄的函数。直接赋值，一代变形虫真的到了10岁。然而问题来了，导致整个游戏逻辑混乱了。

```
struct worm_first worm1;

worm1.name= "alis";
worm1.age= 10;
worm1.length="0.5";
worm1.weight="1";
worm1.sound ="wawawa";
worm1.speed = 1;
```

对于阿猫的代码,铁蛋这样写的, 报错了，说年龄是私有变量不能直接赋值。

```
Worm_first worm1= new Worm_first("tom", 1.1, 2.0,  1, 'huluhulu', 1);
System.out.println("初代变形虫:名字"+worm1.name+",长"+worm1.length+",重"+"worm1.weight"+",虫龄"+worm1.age+
",叫声"+worm1.sound+",爬速"+worm1.speed);
```

这么看来，封装拯救了整个代码逻辑。这一局阿猫完胜。

最后一局

> 阿呆说：我们一代变形虫有个变种，一代狗变形虫，一代狗变形虫是wowo的叫，一代变形虫是momo的叫

阿狗这下感觉有点无力

那我只能在定义一个狗变形虫，赋值叫声是wowo，而一代变形虫赋值叫声叫momo，突然阿狗意识到一个可怕的问题，如果后面又来了二代变形虫，蛇变形虫，鸟变形虫，我到底改怎么办，阿狗陷入了沉思，这下没有开始写代码。

这时候，阿猫一看需求，兴奋了。

多态，这是我期待已久的了。

于是，阿猫把初代变形虫抽象出来，增加了一个抽象的叫声方法,在一代变形虫和狗变形虫中实现不同的叫的声响。而就算之后来了蛇变形虫，鸟变形虫，只要实现相应的叫方法就好了。

```
abstract class Wrom{
   ....
   ....
   abstract void sound();
}

class DogWorm extends Wrom{
   ....
   ....
   public void sound(){
     System.out.println("wowo");
   }
}

class Worm_first extends Wrom{
   .....
   .....
   public void sound(){
      System.out.println("momo");
   }
}
```

而当我们调用时候，不同的变形虫会发出不同的叫声

```
Worm a = new DogWrom(....);
a.sound();

Worm a = new Worm_first(....);
a.sound();
```

最终，使用面向对象思想的阿猫完胜了阿狗，最终获得该项目的负责大权，升职加薪，事业步步高。而阿狗，也不甘落后，回去买了一本书《面向对象葵花宝典》开始细心研读。