1、程序设计追求

- 高内聚：类的内部数据 操作细节自己完成，不允许外部干涉；

- 低耦合：仅对外暴露少量的方法用于使用。

2、封装性的设计思想

隐藏对象内部的复杂性，只对外公开简单的接口。便于外界调用，从而提
高系统的可扩展性、可维护性 。通俗的说 把该隐藏的隐藏起来，该暴露
的暴露出来 。 这就是封装性的设计思想。

3、问题的引入

当创建一个类的对象以后，我们可以通过“对象.属性”的方式，对对象的属性进行赋值。这里，赋值的操作要受到属性的数据类型和储存范围的制约。除此之外，没有其他限制条件。但是在实际问题中，我们

往往在给属性赋值的同时要加入额外的限制条件。这个条件就不能在属性声明时体现，只能通过方法进行限制条件的添加。（比如：set同时，需要避免用户再使用“对象.属性”的方式对属性进行赋值，则需要将属性声明为私有的（private）---》此时，针对属性就体现了封装性）。

4、封装性的体现

Java

中通过将类的属性私有化（private）的同时，提供公共的（public）方法（getXxx ）和（ setXxx ）实现对该属性的操作。

```
class Animal {
 // 将属性legs定义为private ，只能被Animal()类内部访问
 private int legs ;
 public void setLegs( int i ) {
 // 在这里定义方法 eat() 和 move()
 if(i != 0 && i != 2 && i != 4){
 System.out .println ("Wrong number of legs!");
 return;
 }
  legs = i;
  }
  public int getLegs() {
  return legs;
  }

  public void eat(){
  System.out .println ("Eating");
  }
  public void move(){
  System.out .println ("Moving");
  }
}

public class Zoo {
  public static void main(String[] args) {
  Animal animal = new Animal();
  animal.setLegs(2);
```

```
28   System.out.println(animal.getLegs());
29   animal.setLegs(12);
30   animal.eat();
31   animal.move();
32   }
33  }
```

修饰符	类内部	同一个包	不同包的子类	同一个工程
private	Yes			
(缺省)	Yes	Yes		
protected	Yes	Yes	Yes	
public	Yes	Yes	Yes	Yes

7、例题

创建程序在其中定义两个类：Person和PersonTest类。定义如下：用setAge设置人的合法年龄(0~130),用getAge()返回人的年龄。在PersonTest类中实例化Person()类的对象b调用setAge()和getAge()方法，体会Java的封装性。

```
1  /*
2  * 创建程序 在其中定义两个类： Person 和 PersonTest 类 。 定义如下：
3  * 用setAge 设置人的合法年龄(0~130),用getAge()返回人的年龄。在PersonTest类
4  * 中实例化Person()类的对象b调用setAge()和getAge()方法， 体会Java的封装性。
5  */
6
7  public class Person {
8   private int age;
9
10   public int getAge() {
11   return age;
12   }
13
14   public void setAge(int age) {
15   if(age < 0 && age > 130){
16   throw new RuntimeException("转入数据有误！");
```

```
17   }else{
18   this.age = age;
19   }
20   }
21 }
22
23 public class PersonTest {
24   public static void main(String[] args) {
25   Person b = new Person();
26   b.setAge(12);
27   System.out.println("年龄为" + b.getAge());
28   }
29 }
```

8、总结

Java提供了4种权限修饰符来修饰类以及类的内部属性，体现类以及类的内部结构在被调用时可见性的大小范围。