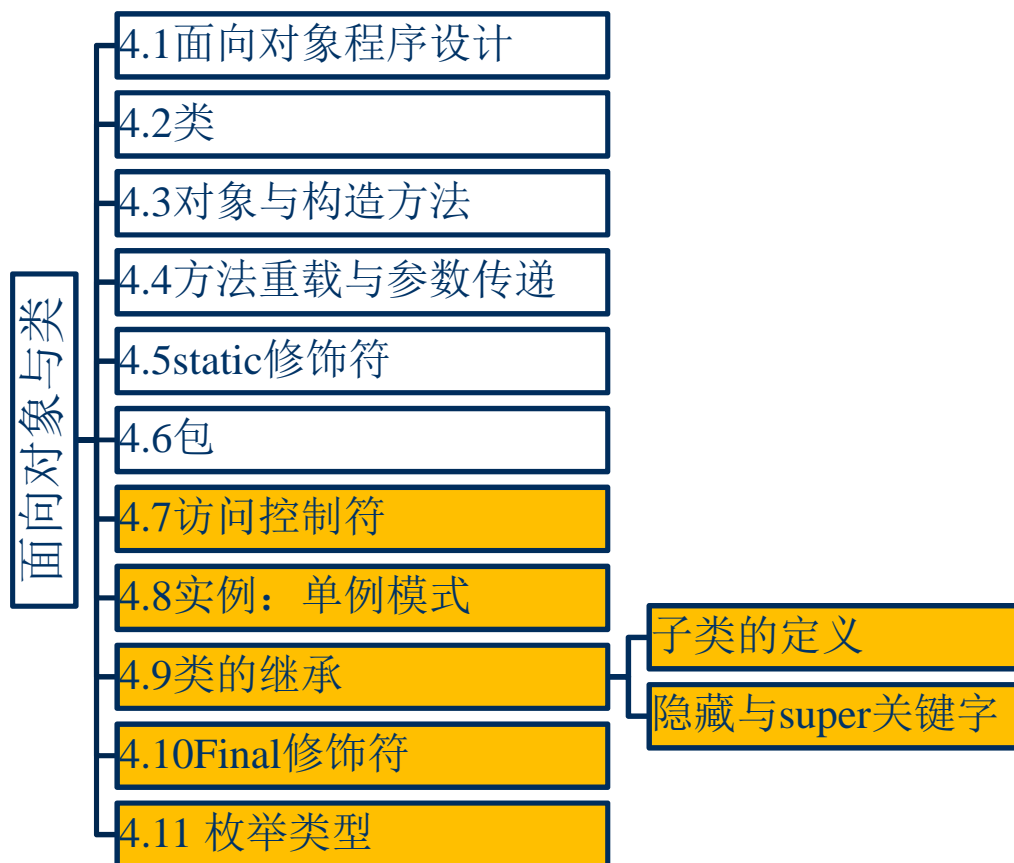




第四章 面向对象与类



第四章：面向对象与类



4.7 访问控制符

- Java提供了四级访问控制权限：公有(public)、受保护(protected)、包权限（默认权限）、私有（private）。
- 权限修饰符主要用来修饰类、数据域和方法。

[访问修饰符][其他修饰符]class 类名 [extends 父类名] [implements 接口列表] {

[访问修饰符][其他修饰符] [成员变量]

[访问修饰符][其他修饰符] [静态初始化块]

[访问修饰符][其他修饰符] [构造方法]

[访问修饰符][其他修饰符] [普通成员方法]

};

修饰符

表 4-2 访问权限修饰符

修饰符	权限级	修饰对象			可见性			
		类	字段	方法	同类	同包	不同包中的子类	不同包中的非子类
public	公有	√	√	√	*	*	*	*
protected	受保护		√	√	*	*	*	
无	包	√	√	√	*	*		
private	私有		√	√	*			

4.7 访问控制符

1、类访问权限修饰符

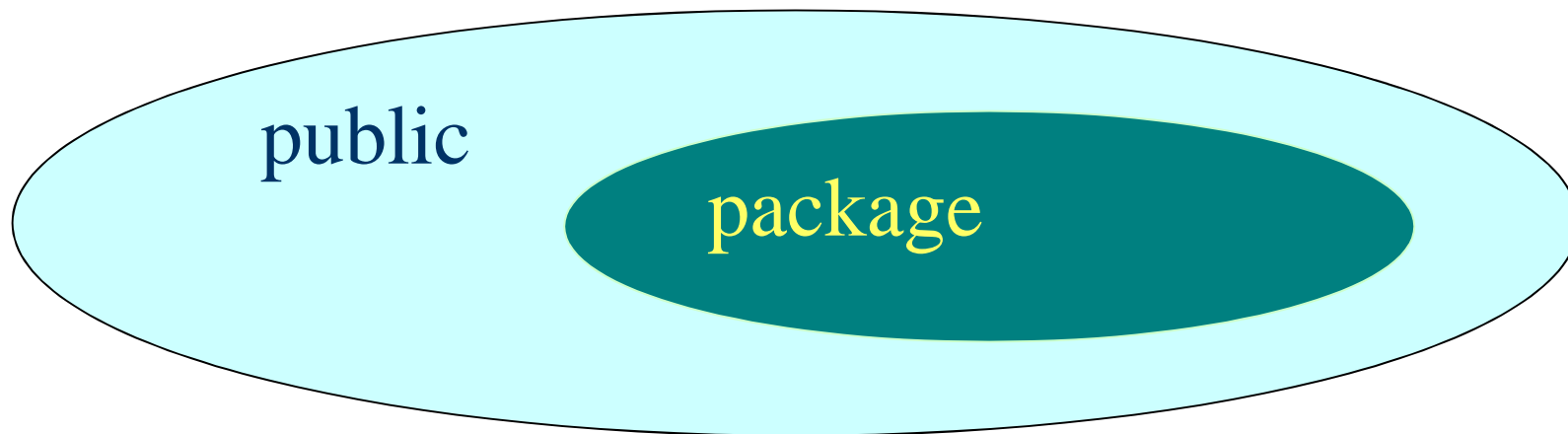
2、成员访问权限修饰符

first step :类访问权限

→ **second step** :成员访问权限

First---类访问权限

类的访问权限有两类：公有（public）、和包权限（无修饰符）



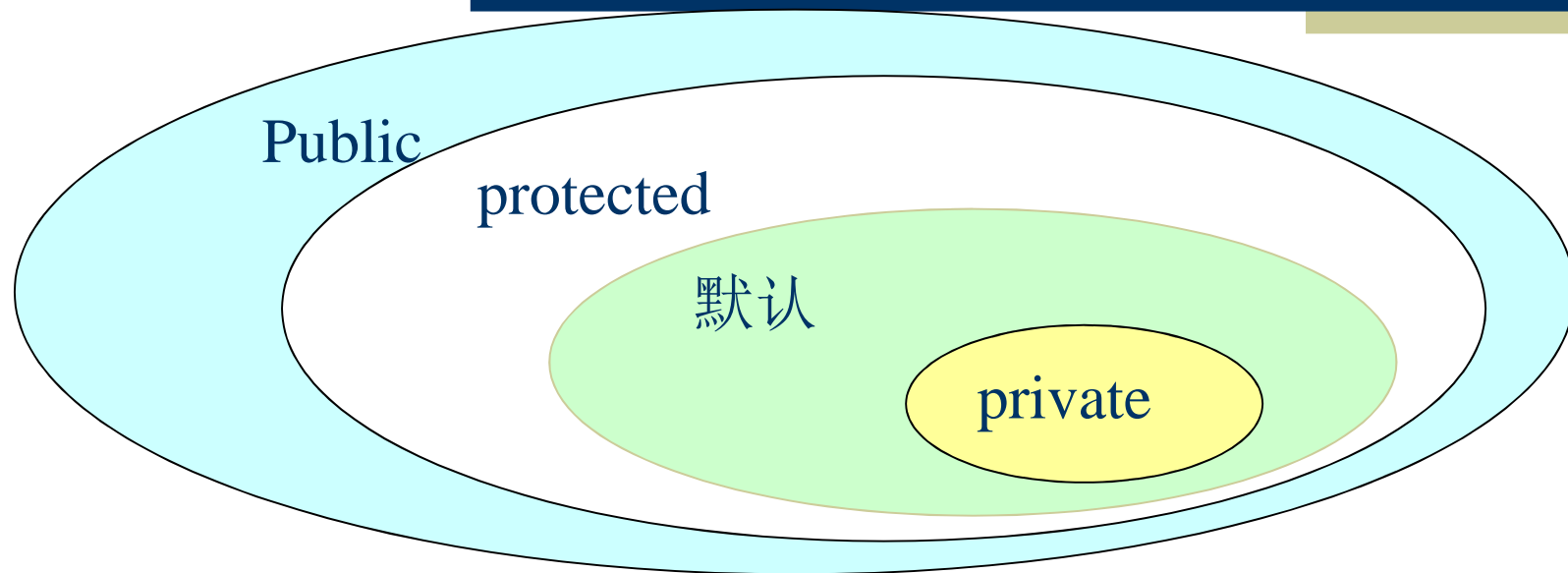
类访问权限


[public] class 类名

	相同的包	不同的包
public	y	y
默认	y	n

- (1) public 类，表示该类对其他类来说是可见的，无论其他类在包内还是包外。
- (2) 无权限修饰符类，表示该类是包内类，对同一个包里的类可见。

Second:成员访问权限



- 
- ◆ 在class类型能被访问的前提下
 - private 成员: 被同类方法访问
 - 默认成员: 被同类和同包内其他类方法访问
 - protected成员: 可被包内类和 子类方法访问
 - public 成员: 如果类型能被访问, 被所有方法访问.

包 p1

MyClass1(公开类)

MyClass2(包内类)

Test测试类

```
package p1;
public class MyClass1{
    public int pub_pub=5;        private int pub_pri=10;
    protected int pub_pro=20;    int pub_defau=30;

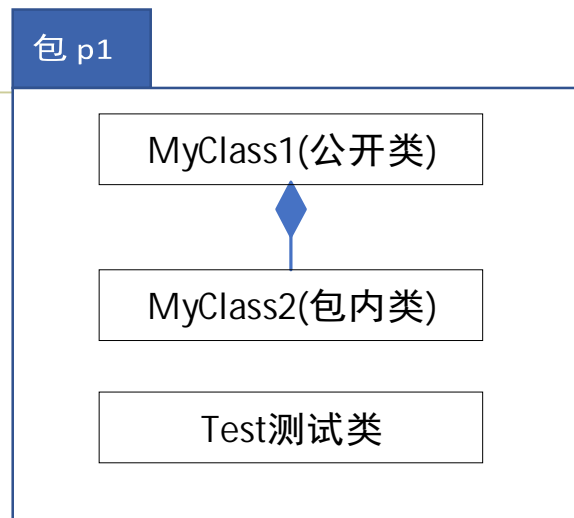
    void inClassAccess(MyClass1 otherMyclass) {
        System.out.printf("pub_pub: %d, pub_pro: %d", pub_pub, pub_pro);
        System.out.printf("pub_defau: %d, pub_pri: %d", pub_defau, pub_pri);
        System.out.println("访问同类对象的属性");

        System.out.printf("otherMyclass. pub_pri: %d", otherMyclass. pub_pri);
    }
}

class MyClass2 extends MyClass1{
    void inSubClassAccess() {
        //同包的子类，可以访问父类非私有的成员
        System.out.println(pub_defau);
        System.out.println(pub_pri); //非法，不可访问
    }
}
```

```
package p1;
public class Test1{
    public static void main(String arg[]){
        //同包非子类，可访问同包类中非私有的成员
        MyClass1 obj1=new MyClass1(); //公有类
        System.out.println(obj1.pub_pub);
        //System.out.println(obj1.pub_pri); //非法，私有成员在类外无法访问。
        System.out.println(obj1.pub_pro);
        System.out.println(obj1.pub_defau);

        MyClass2 obj2=new MyClass2(); //包内类
        obj2.inSubClassAccess();
    }
}
```



```

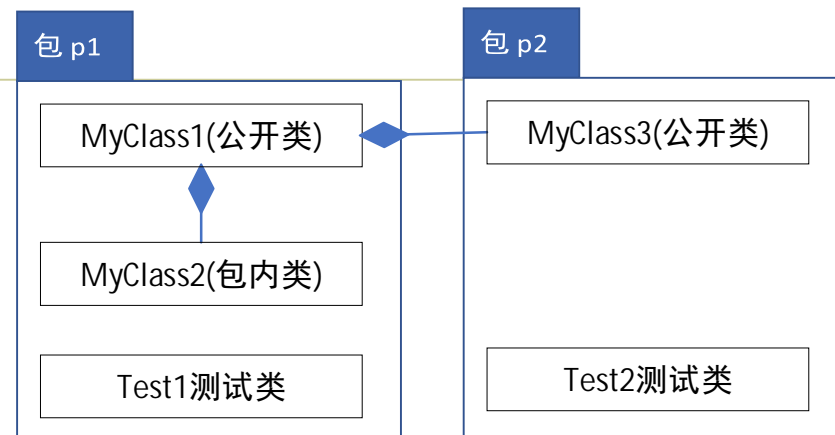
package p2;
import p1.MyClass1;
//import p1.MyClass2; //非法, MyClass2是P1包内类, 对包外类MyClass2不可见
//不同包的子类
class MyClass3 extends MyClass1{
    public void func(MyClass1 superMC, MyClass3 otherC){
        //类只能继承不同包的父类的public、protected成员
        System.out.printf("pub_pub: %d, pub_pro: %d", pub_pub, pub_pro);

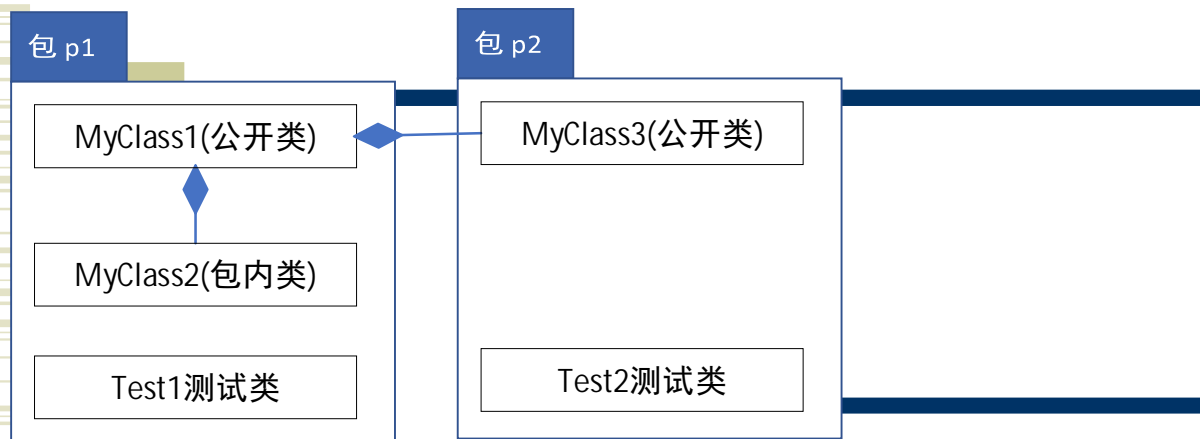
        //无法直接访问继承自父类的私有private成员和包权限成员
        System.out.printf("pub_defau: %d, pub_pri: %d", pub_defau, pub_pri); //非法

        //对同类型的其他对象otherC的成员权限, 等同于对该类型当前对象成员的权限。
        System.out.printf("pub: %d, pro: %d", otherC.pub_pub, otherC.pub_pro);

        //无法访问父类对象的protected权限成员。
        System.out.printf("In Sup Myclass1, pub_pro: %d", superMC.pub_pro); //非法
    }
}

```





//不同包的非子类

```
public class Test2{
    public static void main(String arg[]){
        //不同包的非子类，只能访问其他包的公开类的公开方法
        MyClass1 obj 1=new MyClass1();
        System.out.println(obj 1.pub_pub);
        System.out.println(obj 1.pub_pri); //非法，私有成员对类外方法不可见
        System.out.println(obj 1.pub_pro); //非法，受保护成员对非子类的类外方法不可见
        System.out.println(obj 1.pub_defau); //非法，包权限成员对包外类不可见

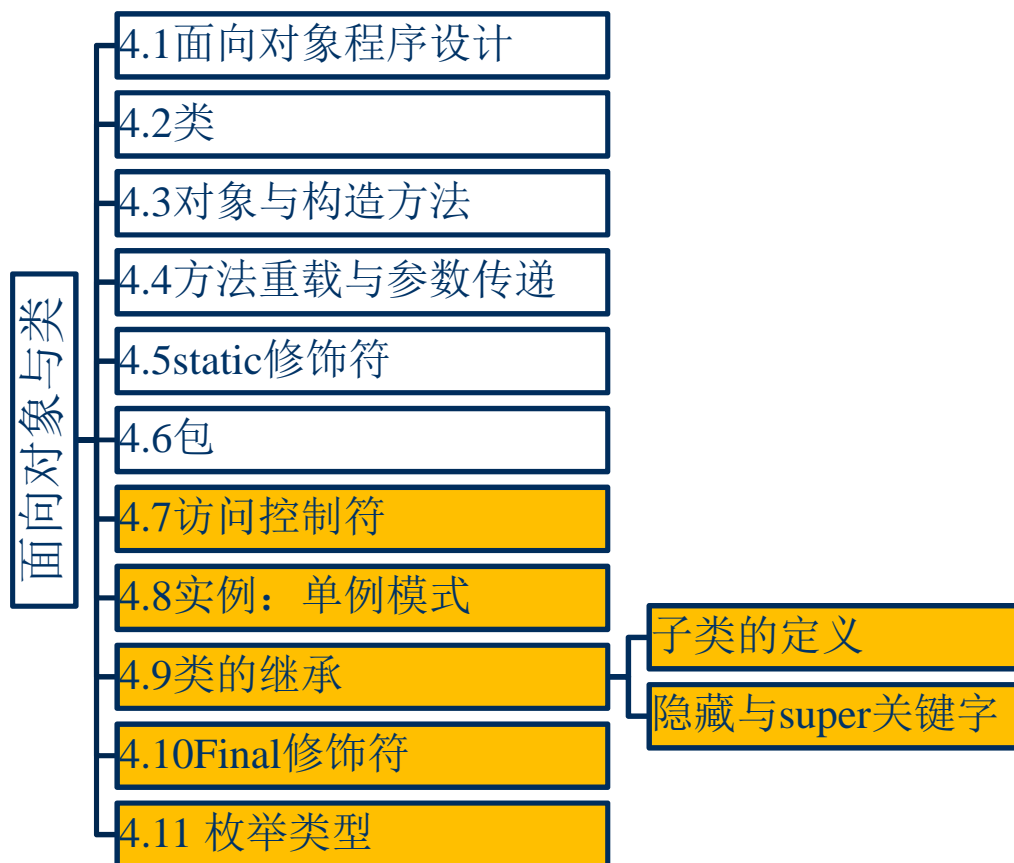
        MyClass2 obj 2=new MyClass2(); //非法，p1中的包内类MyClass2对p2中的类不可见

        //测试MyClass3的func() 方法
        MyClass3 obj 3=new MyClass3();
        MyClass3 otherMyClass3=new MyClass3();
        obj 3.func(obj 1, otherMyClass3);
    }
}
```

访问控制符

- 面向对象设计的松耦合性，需要用到封装，其原则为
 - (1) 将对象的成员变量和实现细节尽量隐藏起来，不允许外部访问
 - (2) 把方法暴露出来，使调用者通过方法对成员变量进行安全访问和操作。

第四章：面向对象与类



4.8 单例/态设计模式

- 构造方法，访问权限封装，static讲解之后，给大家介绍一种**设计模式**。
 - 设计模式是一套关于软件项目的最佳实践与经验的总结。其关注软件在设计层面的问题，与使用的具体语言无关。
- 1、场景：在实际应用中，可能需要整个系统中生成某种类型的**对象不能过多或只有一个**，此时可以使用**单例模式**来实现。
 - 2、定义：所谓单例模式是指一个类有且仅有一个实例向整个系统提供。

4.8 单例/态设计模式

- 单例模式核心思想：

- (1) 构造方法设为私有权限，防止外界任意调用。
- (2) 需要提供一个公有的静态方法获取创建的对象实例。
- (3) 创建的唯一对象是被共享且只能被（2）中的静态方法直接调用，所以，需要将其定义为私有的静态对象。
- (4) 静态对象的初始化可以在类的加载阶段初始化，也可以在外界首次需要对象时在方法（2）中调用构造方法创建对象，之后不再创建对象。

单例模式: 一个类有且仅有一个实例，向整个系统提供。

```
public class Singleton {
    private static Singleton tObj=new Singleton(); //类加载时进行初始化
    private Singleton(){ } //私有构造方法;
    public static Singleton getInstance() { //方法必须被static修饰
        return tObj;
    }
}

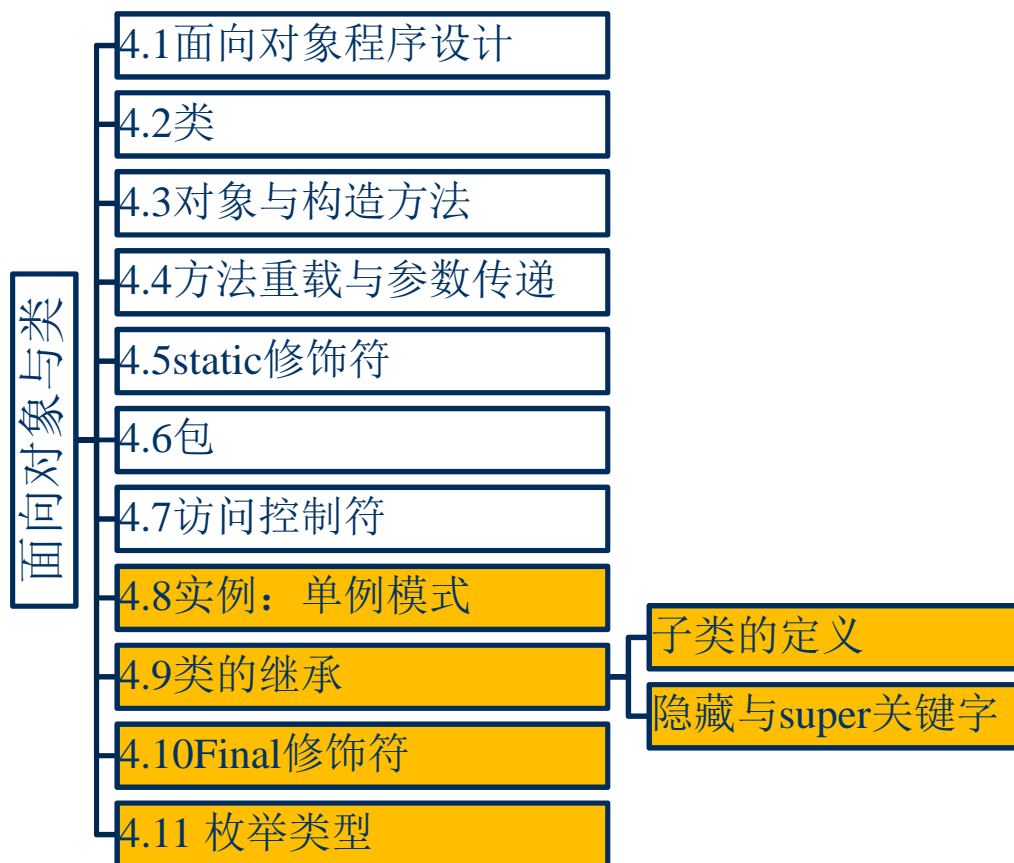
class TestSingleton{
    public static void main(String args[]){
        Singleton t1= Singleton.getInstance();
        Singleton t2= Singleton.getInstance();
        System.out.println(t1==t2); //返回结果为true
    }
}
```

单例模式: 一个类有且仅有一个实例，向整个系统提供。

```
public class Singleton {  
    private static Singleton sObj; //私有且静态  
    private Singleton() { //构造方法必须私有  
    }  
    public static Singleton getInstance() { //方法必须被static修饰  
        if (sObj==null) sObj=new Singleton(); //首次需要时创建  
        return sObj;  
    }  
}  
  
class TestSingleton{  
    public static void main(String args[]){  
        Singleton t1= Singleton.getInstance();  
        Singleton t2= Singleton.getInstance();  
        System.out.println(t1==t2); //返回结果为true  
    }  
}
```

使用单态设计模式在以后的**JAVA**学习中会经常碰到，因为在**JAVA**支持的类库中，大量的采用了此种设计模式。

第四章：面向对象与类



4.9 类的继承

- 继承是面向对象编程的重要特性之一，它是一种由已有的类派生出新类的机制，是实现软件可重用性的一种有效途径。
- 通过继承，子类除了自动拥有父类的属性和行为，同时，子类还可以增加父类所没有的属性和行为，成为一个更特殊的类。继承实际上描述了类之间的“is-a”关系，即子类是一种特殊的父类。

4.9.1 子类的继承

类继承是通过在类的声明中，利用关键字`extends`来说明。语法形式如下：

```
[修饰符] class 子类名 extends 父类名 {  
    //类体  
}
```

说明：

- (1) Java只支持单继承，所以父类只能有一个，默认是Object，Object被称为根类。
- (2) 子类可以继承父类所有属性和方法，但并不意味着总是可以直接访问父类成员。比如，父类中声明为`private`的字段和方法，子类不可见。
- (3) 子类可以添加字段和方法，可以通过定义重名的属性隐藏父类属性（比较少用），也可以通过定义重名方法覆盖父类方法。这部分内容将会在5.2节进一步学习。
- (4) 子类不会自动获得父类的构造方法。例如4.24中，父类Point具有`Point(int,int)`构造方法，不意味着子类ColorPoint具有`ColorPoint(int,int)`构造方法。

```

package chapter5;
class Point{
    private int x,y;
    public Point(){
    }
    public Point(int x,int y){
        this.x=x;
        this.y=y;
        init();
    }
    private void init() {
        System.out.printf("Point(%d,%d)\n", x, y);
    }
    protected void setXY(int x, int y) {
        this.x=x; this.y=y;
    }
    public int getX() {
        return x;
    }
    public int getY() {
        return y;
    }
}

```

【运行结果】

```

Point(2, 2)
ColorPoint<5, 5, white>

```

```

class ColorPoint extends Point{
    String color; //添加字段
    public ColorPoint(int x, int y, String s) {
        color=s;
        init();//非法，因为父类的private方法，对子类不可见
        setXY(x, y); //合法，继承权限的父类成员对子类是可见的。
    }
    public void showInfo() { //添加方法
        System.out.printf("ColorPoint<%d,%d,%s>\n", getX(), getY(), color);
    }
}

public class ExtDemo {
    public static void main(String args[]){
        Point p1=new Point(2, 2);
        ColorPoint cp1=new ColorPoint(5, 5, "white");
        cp1.showInfo();
    }
}

```

4.9.2 隐藏与super关键字

`super` 关键字代表当前对象的父对象引用，可以用来引用父类的成分：

- 父类的构造方法
- 普通方法
- 属性。

4.9.2 隐藏与super关键字

1. super访问父类成员

子类的成员变量或方法如果与父类的成员相同，子类会隐藏同名父类成员。如果子类想访问父类的同名成员，可以使用 **super** 关键字做前缀来访问。
访问形式：

super. 成员变量

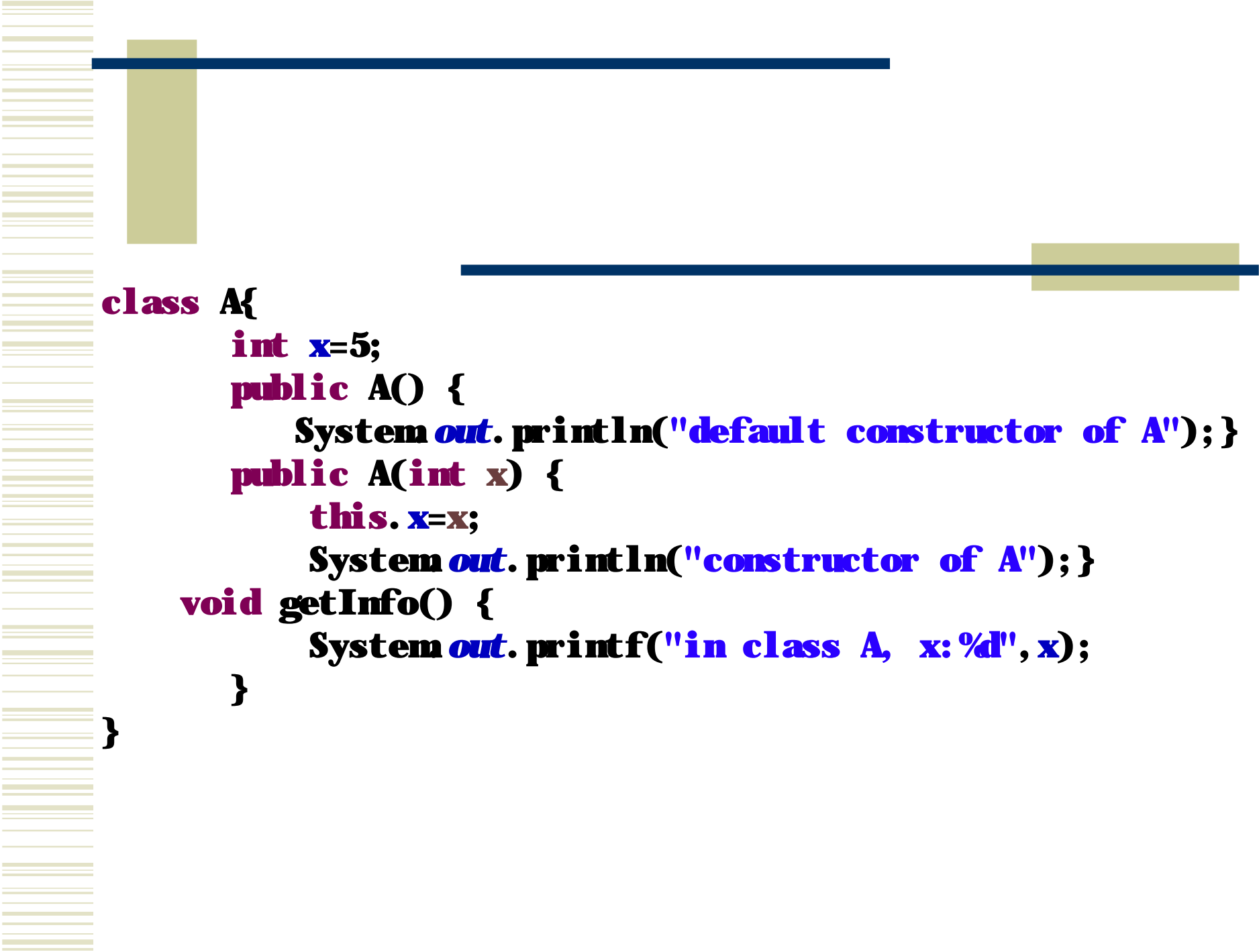
super. 普通方法（[实参]）

4.9.2 隐藏与super关键字

2. 调用父类构造方法

子类不能继承父类的构造方法，子类在构造方法中如果要调用父类的构造方法，可以使用 `super`调用，而且该调用必须是子类构造方法的第一条语句。访问形式：

```
super ([实参])
```



```
class A{
    int x=5;
    public A() {
        System.out.println("default constructor of A");}
    public A(int x) {
        this.x=x;
        System.out.println("constructor of A");}
    void getInfo() {
        System.out.printf("in class A, x:%d", x);
    }
}
```

【运行结果】

constructor of A

constructor of B

in class B,x=10, super.x=15

in class A, x:15

```
class B extends A{
    int x=10;    //与父类同名变量
    public B(int temp) {
        super(temp); //调用父类构造方法，必须放在第一句。
        System.out.println("constructor of B");
    }
    void getInfo() { //调用父类被隐藏变量x
        System.out.printf("in class B, x=%d, super.x=%d", x, super.x);
        super.getInfo(); //调用父类同名方法
    }
}

public class superTest {
    public static void main(String arg[]) {
        B Ref=new B(15);
        Ref.getInfo();
    }
}
```

4.9.2 隐藏与super关键字

3、使用super的注意事项

(1) 通过super不仅可以访问直接父类中定义的属性和方法，还可以访问间接父类中定义的属性和方法。

(2) 由于super指的是对象，所以super不能在static环境中使用，包括类变量、类方法和static语句块。

(3) 在继承关系中，子类的构造方法第一条语句默认为调用父类的无参构造方法（即默认为 `super()`），所以当在父类中定义了有参构造方法，但是没有定义无参构造方法时，编译子类时，会强制要求我们定义一个相同参数类型的构造方法。

4.9.2 隐藏与super关键字

```
class Base1{
    int m=0;
}
class C extends Base1{
    int i;
    public C(int j){i=j;}
}
public class D extends C{
    int i;
    public D(int j1){
        1. //super(6);    //非法, 注释掉后默认调用super(), 父类中没有()。
        2. i=j1+super.m;  //合法, 通过super可调用间接父类的成员
    }
    static void showInfo() {
        3. System.out.println("super.i: "+super.i); //非法, 在static 方法中。
    }
}
```

4.10 final关键字

final 关键字表示：“不可改变，最终的”意思，用于修饰数据、方法和类。

修饰	意义	形式
类	表示最终类，该类不能被继承	[修饰符] final class 类名{ 类体}
方法	表示最终方法，即该方法不能被子类覆盖。	[修饰符] final 返回类型 方法名（[形参列表]）{ 方法体 }
数据	即常量，包括局部变量和字段，一旦赋值就不能再修改。	[修饰符] final 数据类型 变量[=值]

4.10 final关键字

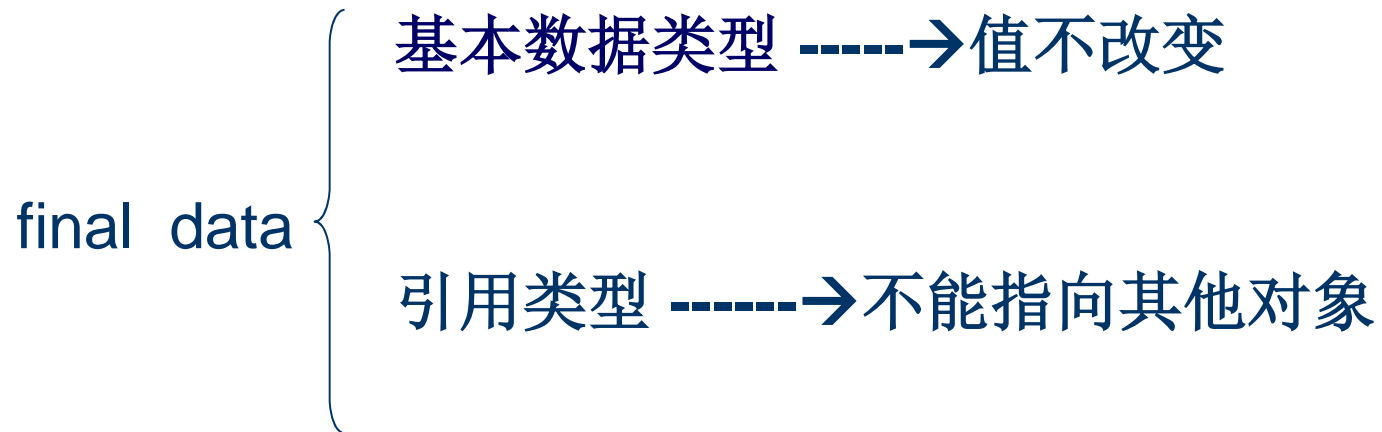
1. final常量

final修饰的常量只能赋值一次，之后不能再修改。它可以是局部常量也可以是成员常量。

- 修饰局部常量时，该常量必须在读取之前被赋值，
- 修饰的是成员常量，则必须在声明时或者在构造方法中被初始化。

4.10 final关键字---final 常量

常量的数据类型分基本数据类型和引用类型。



4.10 final关键字-- final 常量

```
class Value{
    int i=1;
}
public class finalData{
    final int i1=7;
    final Value v2=new Value();

    public void testMethod(){
        final int inSta=1; //局部常量
        inSta++; //非法，已经被赋值，不能再修改
        i1++;    //非法，i1已经被初始化，不能再被修改
        v2.i++;  //合法，v2没有指向新的对象，对原有对象的字段可以进行修改。
        v2=new Value(); //非法，指向新对象，改变了引用值。
    }
}
```

final 常量的几个常见用法

(1) blank finals (空final)

当由同一个类生成的不同对象希望可以有不同的final字段值时，可以在定义该字段时只声明不赋值，通过构造方法对每个对象的final字段进行赋值。

(2) final 参数

final修饰形参，如果修饰的是基本数据类型，表示形参在传入后值不变，如果修饰的是引用类型，表示形参的引用值被赋值后，就不会指向新的对象。

(3) static final 静态常量

是类一级的全局常量，只用于修饰字段而不能用于局部变量，它需要在类型被加载时就完成初始化操作。因此，一定要在定义时或者在static块中就给定初始值。

1、blank finals（空final）

```
class blankFinal {  
    final int j;  
    blankFinal(int x) {  
        j=x;    }  
    public static void main(String[] args) {  
        blankFinal bf1=new blankFinal(5);  
        blankFinal bf2=new blankFinal(6);  
        System.out.println("f in bf1:"+bf1.j); //运行结果 f in bf1:5  
        System.out.println("f in bf2:"+bf2.j); //运行结果 f in bf2:6  
    }  
}
```

(2) final 参数

```
class Go{
}
public class FinalArgument{
    void with( final Go g ){
        g=new Go(); //非法，不能指向新的对象。
    }

    void f(final int i){
        i++; //非法，值不被修改。
    }

    int get(final int i){
        return i+1; //合法，没有修改形参的值
    }
}
```

(3) : 类常量(final static data)

例如:

// (1) 合法, 定义成员常量时初始化

```
static final int i=(int) (Math.random()*26) ;
```

//(2) 先声明, 在类型加载时就初始化, 要在静态块中初始化

```
static final int j;  //  
static {             //  
    j=10;  
}
```

不能进行如下操作:

```
static final int i;    i=10; //非法
```

(3) : 类常量(final static data)

【运行结果】

in static block, m=20
in the constructor,j=20
in the constructor,j=13

```
public class FinalStatic{
    final static int i=10;
    final static int m;
    final int j, k=30;
    static {
        m=20;
        System.out.println("in static block, m="+m);
    } //静态块
    public FinalStatic(){
        j=(int)(Math.random()*26);
        System.out.println("in the constructor,j="+j);
    }
    public static void main(String arg[]){
        FinalStatic t1=new FinalStatic();
        FinalStatic t2=new FinalStatic();
    } }
```

4.10 final关键字---final 类

final 修饰的类不能被继承。防止不必要的方法重写

```
final class SuperClass {  
}
```

```
class SubClass extends SuperClass { //编译错误  
}
```

如java.lang包中的System类、Math类。

4.10 final关键字---final 类

final 修饰的类不能被继承。防止不必要的方法重写

```
final class SuperClass {  
}
```

```
class SubClass extends SuperClass {    //编译错误  
}
```

如java.lang包中的System类、Math类。

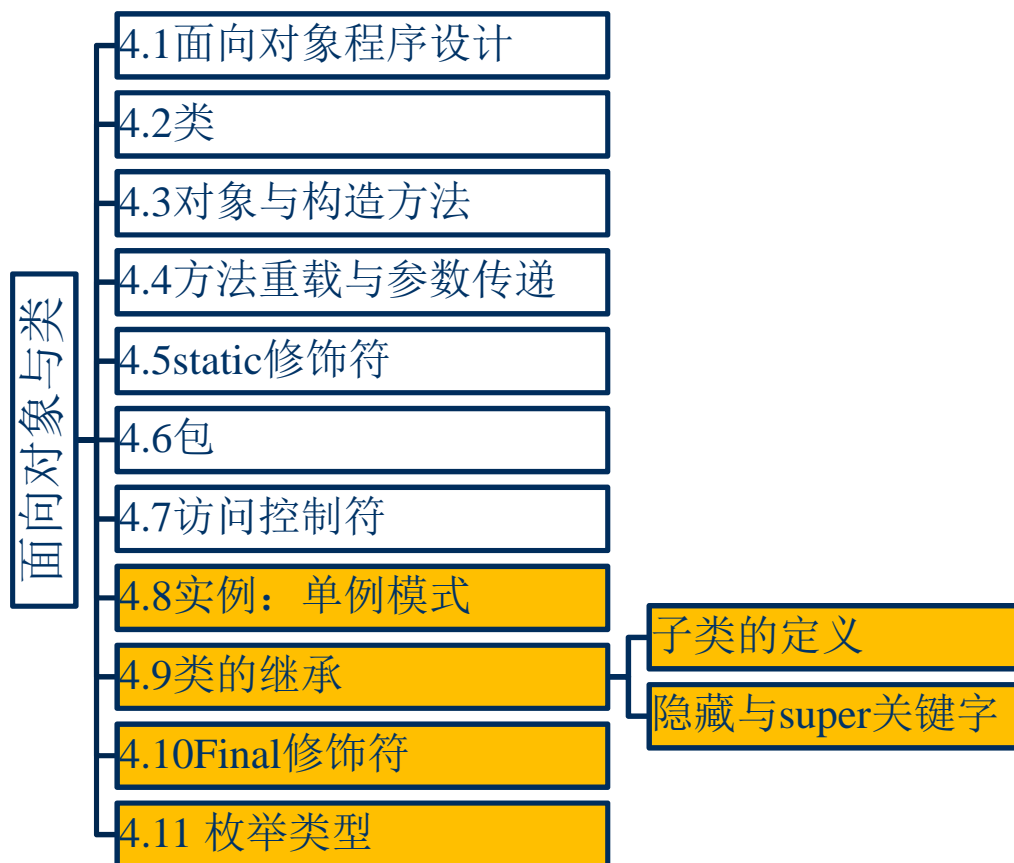
定义final时,请小心

4.10 final关键字---final方法

final方法，是不能被子类所覆盖的方法，说明这种方法提供的功能已经满足当前要求，不需要在子类中对其进行重写。这可以防止子类对父类关键方法的错误重定义。例如：

```
class Base{
    public final void methodA(){}
}
public class son extends Base{
    public final void methodA(){ // 非法，final方法不能被重写。
        System.out.println("test in sub-Class");
    }
}
```

第四章：面向对象与类



4.11 枚举类型

枚举类型实际上是由若干常量构成的集合，其目的是将变量的取值限定在某些常量构成的范围之内，声明为枚举类型的变量取值只能是这些枚举常量中的一个。

1、定义枚举类型

```
[修饰符] enum 枚举类型名 [implements 接口名1, 接口名2, ...]{  
    枚举常量1, 枚举常量2, .....[:]  
    [变量成员声明及初始化;]  
    [方法声明及方法体;]  
}
```

例如：

```
enum Level { //定义枚举类型  
    TOPSECRET, SECRET, CONFIDENTIAL, PUBLIC } //枚举常量
```

4.11 枚举类型

说明：

- (1) 当使用枚举类型成员时，直接使用枚举名称调用成员即可。
- (2) 枚举类型实质上是继承`java.lang.Enum`的类，每一个枚举值都可以看作是类的实例。
- (3) 枚举值是`public`、`static`、`final`的，且会被分配一个`int`型从0开始的序号。
- (4) 当枚举类型包含其他变量和方法时，最后一个枚举常量后的分号不能省略。

4.11 枚举类型

2、常用方法

(1) 在编译时，编译器会自动塞进一个静态方法`values()`，该方法返回所有枚举常量构成的数组。该方法通常与for-each结构结合使用，用来遍历一个枚举类型的值。

(2) 枚举类型也继承了来自父类Enum的方法，如常见的方法：

- `final String name()` //获取枚举常量对应的字符串
- `final int ordinal()` //获取枚举成员的索引位置

4.11 枚举类型

```
enum Level {  
    TOPSECRET,  
    SECRET,  
    CONFIDENTIAL,  
    PUBLIC  
}  
  
public class Enum2{  
    public static void main(String arg[]) {  
        for(Level lc:Level.values()) {  
            int order=lc.ordinal();  
            switch(lc) {  
                case TOPSECRET: System.out.println(order+" >=85.0");  
                    break;  
                case SECRET: System.out.println(order+" >=70.0&&<85");  
                    break;  
                case CONFIDENTIAL: System.out.println(order+" >=60.0&&<70");  
                    break;  
                case PUBLIC: System.out.println(order+" <60");  
            }  
        }  
    }  
}
```

4.11 枚举类型

3、自定义属性和方法

- 既然枚举类型本质上是类，那就可以在枚举常量以外定义数据和方法，用于补充枚举常量除名称和序号以外的信息。
- 此外枚举类型可以定义构造方法，但枚举类型的构造方法只能是 `private` 的，且默认是 `private` 的，这样可以防止调用者自行定义枚举类型的对象。

4.11 枚举类型

```
enum Level2 {  
    TOPSECRET(">=85.0"),  
    SECRET(">=70.0&&<85"),  
    CONFIDENTIAL(">=60.0&&<70"),  
    PUBLIC;  
    private String score;  
    private Level2() {score="<60";}   
    private Level2(String d1) {score=d1;}  
    String getScore() {return score;}  
    void setScore(String d1) {score=d1;}  
}
```

```
public class Enum3 {  
    public static void main(String arg[]) {  
        for(Level2 lc:Level2.values())  
            System.out.printf("%-4d%-14s%-14s\n", lc.ordinal(), lc.name(),  
lc.getScore());  
    }  
}
```

【运行结果】

```
0 >=85.0  
1 >=70.0&&<85  
2 >=60.0&&<70  
3 <60
```