* [author](https://github.com/3293172751)
* ![fig:](data:image/svg+xml;charset=utf-8;base64,)

# 第9节 java面对对象补充

* [回到目录](../README.md)
* [回到项目首页](../../README.md)
* [上一节](8.md)

❤️💕💕java的学习指南,从入门到大师篇章。Myblog:[http://nsddd.top](http://nsddd.top/)

[第9节 java面对对象补充](#第9节-java面对对象补充)  
 [super关键字](#super关键字)  
 [Java多态](#java多态)  
 [java内部类](#java内部类)  
 [public内部](#public内部)  
 [静态内部类](#静态内部类)  
 [从内部类访问外部类](#从内部类访问外部类)  
 [抽象类和方法](#抽象类和方法)  
 [上转型](#上转型)  
 [为什么以及何时使用抽象类和方法？](#为什么以及何时使用抽象类和方法)  
 [接口](#接口)  
 [接口注意事项：](#接口注意事项)  
 [为什么以及何时使用接口？](#为什么以及何时使用接口)  
 [接口支持上转型](#接口支持上转型)  
 [多个接口](#多个接口)  
 [多接口上转型](#多接口上转型)  
 [枚举](#枚举)  
 [类中的枚举](#类中的枚举)  
 [Switch 语句中的枚举](#switch-语句中的枚举)  
 [枚举和类之间的区别](#枚举和类之间的区别)  
 [为什么以及何时使用枚举？](#为什么以及何时使用枚举)  
 [END 链接](#end-链接)

## super关键字

⚡super代表父类的引用，用于访问父类属性、方法和构造器

## Java多态

多态意味着“多种形式”，当我们有许多通过继承相互关联的类时就会发生这种情况。

就像我们在上面继承中中指定的那样； **继承**让我们从另一个类继承属性和方法。**多态性** 使用这些方法来执行不同的任务。这使我们能够以不同的方式执行单个操作。

class Animal {  
 public void animalSound() {  
 System.out.println("The animal makes a sound");  
 }  
}  
  
class Pig extends Animal {  
 public void animalSound() {  
 System.out.println("The pig says: wee wee");  
 }  
}  
  
class Dog extends Animal {  
 public void animalSound() {  
 System.out.println("The dog says: bow wow");  
 }  
}

请记住，在继承中，我们使用extends关键字从类继承。

现在我们可以创建Pig和Dog对象并调用animalSound()它们的方法：

**例子：**

class Animal {  
 public void animalSound() {  
 System.out.println("The animal makes a sound");  
 }  
}  
  
class Pig extends Animal {  
 public void animalSound() {  
 System.out.println("The pig says: wee wee");  
 }  
}  
  
class Dog extends Animal {  
 public void animalSound() {  
 System.out.println("The dog says: bow wow");  
 }  
}  
  
class Main {  
 public static void main(String[] args) {  
 Animal myAnimal = new Animal(); // Create a Animal object  
 Animal myPig = new Pig(); // Create a Pig object  
 Animal myDog = new Dog(); // Create a Dog object  
 myAnimal.animalSound();  
 myPig.animalSound();  
 myDog.animalSound();  
 }  
}

**编译：**

The animal makes a sound  
The pig says: wee wee  
The dog says: bow wow

## java内部类

在 Java 中，也可以嵌套类（类中的类）。嵌套类的目的是将属于一起的类分组，这使您的代码更具可读性和可维护性。

要访问内部类，先创建外部类的对象，然后再创建内部类的对象：

**例子：**

class OuterClass {  
 int x = 10;  
  
 class InnerClass {  
 int y = 5;  
 }  
 class Class2{  
 int y = 10;  
 }  
}  
  
public class Main {  
 public static void main(String[] args) {  
 OuterClass myOuter = new OuterClass();  
 OuterClass.InnerClass myInner = myOuter.new InnerClass();  
 OuterClass.Class2 myInner2 = myOuter.new Class2();  
 System.out.println(myInner.y + myOuter.x); //15  
 System.out.println(myInner2.y + myOuter.x); //20  
 }  
}

### public内部

与“常规”类不同，内部类可以是private或protected。如果您不希望外部对象访问内部类，请将类声明为private：

class OuterClass {  
 int x = 10;  
   
 private class InnerClass {  
 int y = 5;  
 } //不能被外部对象访问  
}  
  
public class Main {  
 public static void main(String[] args) {  
 OuterClass myOuter = new OuterClass();  
 OuterClass.InnerClass myInner = myOuter.new InnerClass();  
 System.out.println(myInner.y + myOuter.x); //myInner.y ×  
 }  
}

如果您尝试从外部类访问私有内部类，则会发生错误：

Main.java:13: error: OuterClass.InnerClass has private access in OuterClass OuterClass.InnerClass myInner = myOuter.new InnerClass(); ^

### 静态内部类

内部类也可以是static，这意味着您可以在不创建外部类的对象的情况下访问它：

class OuterClass {  
 int x = 10;  
  
 static class InnerClass {  
 int y = 5;  
 }  
}  
  
public class Main {  
 public static void main(String[] args) {  
 OuterClass.InnerClass myInner = new OuterClass.InnerClass();  
 System.out.println(myInner.y);  
 }  
}  
  
// Outputs 5

**注意：**就像static属性和方法一样，static内部类不能访问外部类的成员。

## 从内部类访问外部类

内部类的一个优点是它们可以访问外部类的属性和方法：

**例子：**

class OuterClass {  
 int x = 10;  
  
 class InnerClass {  
 public int myInnerMethod() {  
 return x;  
 }  
 }  
}  
  
public class Main {  
 public static void main(String[] args) {  
 OuterClass myOuter = new OuterClass();  
 OuterClass.InnerClass myInner = myOuter.new InnerClass();  
 System.out.println(myInner.myInnerMethod());  
 }  
}  
// Outputs 10

## 抽象类和方法

数据**抽象**是隐藏某些细节并仅向用户显示基本信息的过程。  
抽象可以通过**抽象类**或 **接口**来实现。

abstract关键字是非访问修饰符，用于类和方法 ：

* **抽象类：**是一个受限制的类，不能用于创建对象（要访问它，必须从另一个类继承）。
* **抽象方法：**只能在抽象类中使用，它没有主体。主体由子类（继承自）提供。

抽象类可以同时具有抽象方法和常规方法：

abstract class Animal {  
 public abstract void animalSound();  
 public void sleep() {  
 System.out.println("Zzz");  
 }  
}

从上面的示例中，无法创建 Animal 类的对象：

Animal myObj = new Animal(); //将产生一个错误

**要访问抽象类，它必须从另一个类继承。**让我们将我们在[多态](https://www.w3schools.com/java/java_polymorphism.asp)一章中使用的 Animal 类转换为抽象类：

请记住，**在继承中**，我们使用extends关键字从类继承。

// Abstract class  
abstract class Animal {  
 // Abstract method (does not have a body)  
 public abstract void animalSound();  
 // Regular method  
 public void sleep() {  
 System.out.println("Zzz");  
 }  
}  
  
// Subclass (inherit from Animal)  
class Pig extends Animal {  
 public void animalSound() {  
 // The body of animalSound() is provided here  
 System.out.println("The pig says: wee wee");  
 }  
}  
  
class Main {  
 public static void main(String[] args) {  
 Pig myPig = new Pig(); // Create a Pig object  
 myPig.animalSound();  
 myPig.sleep();  
 }  
}

### 上转型

// Abstract class  
abstract class Animal {  
 // Abstract method (does not have a body)  
 public abstract void animalSound();  
 // Regular method  
 public void sleep() {  
 System.out.println("Zzz");  
 }  
}  
  
// Subclass (inherit from Animal)  
class Pig extends Animal {  
 public void animalSound() {  
 // The body of animalSound() is provided here  
 System.out.println("The pig says: wee wee");  
 }  
}  
  
class Main {  
 public static void main(String[] args) {  
 Pig myPig = new Pig(); // Create a Pig object  
 //Animal animal = new Animal(); //不能定义  
 //animal.sleep();  
   
 Animal myPig2 = new Pig(); //上转型  
 myPig.animalSound();  
 myPig.sleep();  
 myPig2.animalSound();  
 myPig2.sleep();  
 }  
}

### 为什么以及何时使用抽象类和方法？

实现安全性 - 隐藏某些细节并仅显示对象的重要细节。

**注意：**抽象也可以通过接口来实现，您将在下一章中了解更多信息。

## 接口

在 Java**中实现抽象的另一种方法是使用接口**。

Aninterface是一个完全“**抽象类**”，用于将相关方法与空主体分组：

// interface  
interface Animal {  
 public void animalSound(); // interface method (does not have a body)  
 public void run(); // interface method (does not have a body)  
}

要访问接口方法，接口必须由另一个使用implements 关键字（而不是）的类“实现”（有点像继承extends）。接口方法的主体由“实现”类提供：

// Interface  
interface Animal {  
 public void animalSound(); // interface method (does not have a body)  
 public void sleep(); // interface method (does not have a body)  
}  
  
// Pig "implements" the Animal interface  
class Pig implements Animal {  
 public void animalSound() {  
 // The body of animalSound() is provided here  
 System.out.println("The pig says: wee wee");  
 }  
 public void sleep() {  
 // The body of sleep() is provided here  
 System.out.println("Zzz");  
 }  
}  
  
class Main {  
 public static void main(String[] args) {  
 Pig myPig = new Pig(); // Create a Pig object  
 myPig.animalSound();  
 myPig.sleep();  
 }  
}

### 接口注意事项：

* 与**抽象类**一样，接口**不能**用于创建对象（在上面的示例中，无法在 MyMainClass 中创建“Animal”对象）
* 接口方法没有主体——主体由“实现”类提供
* 在实现接口时，您必须覆盖其所有方法
* 接口方法默认abstract是 public
* 接口属性是默认的public， static并且final
* 接口不能包含构造函数（因为它不能用于创建对象）

### 为什么以及何时使用接口？

1）为了实现安全——隐藏某些细节，只显示一个对象（接口）的重要细节。

2）Java不支持“多重继承”（一个类只能从一个超类继承）。但是，它可以用接口来实现，因为类可以**实现**多个接口。 **注意：**要实现多个接口，请用逗号分隔它们（参见下面的示例）。

### 接口支持上转型

interface Animal {  
 public void animalSound(); // interface method (does not have a body)  
 public void sleep(); // interface method (does not have a body)  
}  
  
class Pig implements Animal {  
 public void animalSound() {  
 System.out.println("The pig says: wee wee");  
 }  
 public void sleep() {  
 System.out.println("Zzz");  
 }  
}  
  
class Main {  
 public static void main(String[] args) {  
 Pig myPig = new Pig();  
 myPig.animalSound();  
 myPig.sleep();  
   
 Animal myPig2 = new Pig(); //上转型变量  
 myPig2.animalSound();  
 myPig.sleep();  
 }  
}

### 多个接口

要实现多个接口，请用逗号分隔它们：

interface FirstInterface {  
 public void myMethod(); // interface method  
}  
  
interface SecondInterface {  
 public void myOtherMethod(); // interface method  
}  
  
class DemoClass implements FirstInterface, SecondInterface {  
 public void myMethod() {  
 System.out.println("Some text..");  
 }  
 public void myOtherMethod() {  
 System.out.println("Some other text...");  
 }  
}  
  
class Main {  
 public static void main(String[] args) {  
 DemoClass myObj = new DemoClass();  
 myObj.myMethod();  
 myObj.myOtherMethod();  
 }  
}

### 多接口上转型

/\*  
 \* @Description:   
 \* @Author: xiongxinwei 3293172751nss@gmail.com  
 \* @Date: 2022-09-08 14:35:45  
 \* @LastEditTime: 2022-09-08 14:36:22  
 \* @FilePath: \code\top.nsddd.erer\src\a\a.java  
 \* @blog: https://nsddd.top  
 \*/  
interface FirstInterface {  
 public void myMethod(); // interface method  
}  
  
interface SecondInterface {  
 public void myOtherMethod(); // interface method  
 public void a(int a);   
}  
  
// DemoClass "implements" FirstInterface and SecondInterface  
class DemoClass implements FirstInterface, SecondInterface {  
 public void myMethod() {  
 System.out.println("Some text..");  
 }  
 public void myOtherMethod() {  
 System.out.println("Some other text...");  
 }  
 public void a(int a){  
 System.out.println("A = "+a);  
 }   
}  
  
class a {  
 public static void main(String[] args) {  
 DemoClass myObj = new DemoClass();  
 myObj.myMethod(); //Some text..  
 myObj.myOtherMethod(); //Some other text...  
   
 //上转型  
 FirstInterface my1 = new DemoClass();  
 my1.myMethod(); //Some text..  
 //my1.myOtherMethod(); FirstInterfacec创建的上转型不能用SecondInterface接口方法   
   
 SecondInterface my2 = new DemoClass();  
 //my2.myMethod(); SecondInterface创建的上转型不能用 FirstInterfacec接口方法   
 my2.myOtherMethod(); //SecondInterface  
 my2.a(1000); //1000  
 }  
}

## 枚举

Anenum是一个特殊的“类”，它表示一组 **常量**（不可更改的变量，如final变量）。

要创建enum，请使用enum关键字（而不是类或接口），并用逗号分隔常量。请注意，**它们应该是大写字母**：

enum Level {  
 LOW,  
 MEDIUM,  
 HIGH  
}

您可以使用**点**语法访问enum常量：

Level myVar = Level.MEDIUM;

**Enum**是“枚举”的缩写，意思是“特别列出”。

## 类中的枚举

你也可以enum在一个类里面：

public class Main {  
 enum Level {  
 LOW,  
 MEDIUM,  
 HIGH  
 }  
  
 public static void main(String[] args) {  
 Level myVar = Level.MEDIUM;   
 System.out.println(myVar);  
 }  
}

输出将是：

MEDIUM

### Switch 语句中的枚举

枚举经常在switch语句中用于检查相应的值：

enum Level {  
 LOW,  
 MEDIUM,  
 HIGH  
}  
  
public class Main {  
 public static void main(String[] args) {  
 Level myVar = Level.MEDIUM;  
  
 switch(myVar) {  
 case LOW:  
 System.out.println("Low level");  
 break;  
 case MEDIUM:  
 System.out.println("Medium level");  
 break;  
 case HIGH:  
 System.out.println("High level");  
 break;  
 }  
 }  
}

输出将是：

Medium level

enum 类型有一个values()方法，它返回一个包含所有枚举常量的数组。当您想要遍历枚举的常量时，此方法很有用：

enum Level {  
 LOW,  
 MEDIUM,  
 HIGH  
}  
  
public class Main {   
 public static void main(String[] args) {  
 int i = 1;  
 for (Level myVar : Level.values()) {  
 System.out.println("myVar中的第"+(i++)+"的值为:"+myVar);  
 }  
 }   
}

输出将是：

myVar中的第1的值为:LOW  
myVar中的第2的值为:MEDIUM  
myVar中的第3的值为:HIGH

### 枚举和类之间的区别

一个enum罐头，就像一个class，有属性和方法。唯一的区别是枚举常量是public,static和final (不可更改 - 不能被覆盖)。

Anenum不能用于创建对象，也不能扩展其他类（但它可以实现接口）。

### 为什么以及何时使用枚举？

当你有你知道不会改变的值时使用枚举，比如月日、日、颜色、卡片组等。

## END 链接

* [回到目录](../README.md)
* [上一节](8.md)
* [下一节](10.md)
* [参与贡献❤️💕💕](https://github.com/3293172751/Block_Chain/blob/master/Git/git-contributor.md)