

笔记总链接:http://bbs.itheima.com/thread-200600-1-1.html

非贷款,0元入学,不1万就业不给1分钱学费,我们已干四年了!

4、继承

4.7 接口

接口应用综合案例

代码: /* 01. 02.

笔记本电脑使用。 为了扩展笔记本的功能,但日后出现什么功能设备不知道。

03. 04.

因此需要定义一个规则,只要日后出现的设备都符合这个规则就可以了。 规则在java中就是接口。 06.

05.

interface USB{//暴露的原则 public void open(); public void close(); }

07. 08. 09. //实现原则

10. 11. 12. //这些设备和电脑的耦合性降低了 13. 14. class UPan implements USB{ public void open(){ System.out.println("upan open");

}

15. 16. 17. public void close(){ 18. System.out.println("upan close"); 19. 20. } 21. } 22. class UsbMouse implements USB{ 23. 24. public void open(){ System.out.println("usbMouse open"); 25. 26. 27. public void close(){ System.out.println("usbMouse close"); 28. 29. }

} 30. 31. 32. class BookPC{ public static void main(String[] args){ //功能扩展了 useUSB(new UPan()); } //使用原则 if(u != null){ u.open(); u.close(); } }

33. 34. 35. 36. 37. public static void useUSB(USB u){//接口类型的引用,用于接收(指向)接口的子类对 38. 39. 40. 41. 42. 43. } 44. 复制代码 运行结果: - - X 画 管理员: C:\Windows\system32\cmd.exe D:\code\day10>javac BookPC.java Ξ D:\code\day10>java BookPC upan open upan close

4.8 多态

例:动物中猫,狗。 猫这个对象对应的类型是猫类型:猫 x = new 猫(); 同时猫也是动物中的一种,也可以把猫称为动物:动物 y = new 猫(); 动物是猫和狗具体事物中抽取出来的父类型。

前提:

定义:某一类事物的多种存在形态。

父类型引用指向了子类对象。

多态性简单说就是一个对象对应着不同类型。 体现: 父类或者接口的引用指向或者接收自己的子类对象。 作用: 多态的存在提高了程序的扩展性和后期可维护性。

1. 需要存在继承或者实现关系。 2. 需要有覆盖操作。 好处: 提高了代码的扩展性,前期定义的代码可以使用后期的内容。 弊端:

前期定义的内容不能使用(调用)后期子类的特有内容。 示例1: abstract class Animal{ 01. 02. abstract void eat(); 03. 04.

class Dog extends Animal{ 05. void eat(){ 06. System.out.println("啃骨头"); 07. 08. void lookHome(){ 09. System.out.println("看家"); 10. } 11. } 12. 13. 14. class Cat extends Animal(void eat(){ 15. System.out.println("吃鱼"); 16. 17.

void catchMouse(){

class Pig extends Animal{

void eat(){

void gongdi(){

class DuoTaiDemo{

}

System.out.println("抓老鼠");

System.out.println("饲料");

System.out.println("拱地");

public static void main(String[] args){

Cat c = new Cat();

Dog d = new Dog();

18.

19. 20. 21. 22. 23.

24.

25. 26.

27.

28. 29.

30. 31. 32.

33.

34. 35.

36.

09.

10.

11.

12.

13. 14.

15. 16.

17. 18.

19. 20.

21.

22.

23.

运行结果:

示例3:

01.

02.

03.

04.

05.

06. 07.

09. 10. */

}

}

复制代码

37. method(c); method(d); 38. method(new Pig()); 39. 40. } 41. public static void method(Animal a){ 42. a.eat(); 43. } 44. 45. 复制代码 运行结果: - 0 X ■ 管理员: C:\Windows\system32\cmd.exe D:\code\day10>javac DuoTaiDemo.java Ξ D:∖code∖day10>java DuoTaiDemo 示例2: class DuoTaiDemo{ 01. public static void main(String[] args){ 02. //自动类型提升,猫对象提升到了动物类型。但是特有功能无法访问,作用就是限制对特有功能 03. 的访问。 //专业讲: 向上转型,将子类型隐藏。就不能使用子类的特有方法了。 04. 05. Animal a = new Cat(); a.eat(); 06. //a.catchMouse();//报错 07. 08. //如果还想用具体动物猫的特有功能。

Cat c = (Cat)a; //向下转型的目的是为了能够使用子类中的特有方法。

//Cat c = (Cat)a;//但是类型不能随意转换,否则可能会报出ClassCastException的异常

_ 0 X

=

//注意:对于转型,自始至终都是子类对象在做类型的变化。

//你可以将该对象进行向下转型。

//Animal a = new Dog();

public static void method(Animal a){

c.eat();

a.eat();

画 管理员: C:\Windows\system32\cmd.exe

D:∖code∖day10>java DuoTaiDemo

毕老师和毕姥爷的故事

void 讲课(){

void 钓鱼(){

void 讲课(){

void 看电影(){

}

class DuoTaiDemo{

class DuoTaiDemo{

public static void main(String[] args){

public static void method(Animal a){

if(a instanceof Cat){

Cat c = (Cat)a;

c.catchMouse();

else if (a instanceof Dog){

编译时:参考引用型变量所属的类中是否有调用的成员变量。有,编译通过,没有,编译失败。

运行时:参考引用型变量所属的类中是否有调用的成员变量,并运行该所属类中的成员变量。

Dog d = (Dog)a;d.lookHome();

a.eat();

}

}

简单说:编译和运行都参考等号的左边。

int num = 3;

class Zi extends Fu{

int num = 4;

}

01.

02. 03. 04. 05.

06. 07. 08.

09.

10. 11.

12. 13.

14. 15.

16.

17.

}

复制代码

多态时,成员的特点:

class Fu{

1. 成员变量

示例:

01.

02.

03. 04. 05.

06. 07. 08.

14.

15. 16.

17.

18.

运行结果:

zi show

3. 静态函数

示例:

01.

02.

03. 04. 05. 06.

07.

08.

09.

10. 11.

12.

}

}

复制代码

运行结果:

13. 14. System.out.println("管理");

System.out.println("钓鱼");

System.out.println("Java");

System.out.println("看电影");

public static void main(String[] args){

毕老师 x = new 毕老师();

毕姥爷 y = new 毕老师();

x.讲课(); //Java

y. 讲课(); //Java

y.钓鱼(); //钓鱼

毕老师 z = (毕老师)y;

x.看电影(); //看电影

class 毕姥爷{

}

D:\code\day10>javac DuoTaiDemo.java

c.catchMouse();

} 11. 12. class 毕老师 extends 毕姥爷{ 13. 14. 15. 16. 17. 18.

19.

20.

21. 22.

23.

24.

25.

26. 27.

29.

30. 31.

32.

}

z.看电影(); //看电影 33. 34. } 35. } 复制代码 运行结果: _ _ _ X om 管理员: C:\Windows\system32\cmd.exe D:∖code∖day10>javac DuoTaiDemo.java D:∖code∖day10>java DuoTaiDemo Ξ Java 看电影 P.S. instanceof : 用于判断对象的具体类型,只能用于引用数据类型判断,通常在向下转型前用于健壮性的判 断。 示例4:

09. class DuoTaiDemo{ public static void main(String[] args){ 10. Zi f1 = new Zi(); 11. System.out.println(f1.num); 12. Fu f2 = new Zi(); 13. 14. System.out.println(f2.num); 15. } 16. 复制代码 运行结果: - 0 X 画 管理员: C:\Windows\system32\cmd.exe D:∖code∖day10>javac DuoTaiDemo.java D:\code\day10>java DuoTaiDemo 2. 成员函数(非静态) 编译时:参考引用型变量所属的类中是否有调用的函数。有,编译通过。没有,编译失败。 运行时:参考的是对象所属的类中是否有调用的函数。 简单说:编译看左边,运行看右边。 示例: class Fu{ 01. void show(){ 02. System.out.println("fu show"); 03. } 04. 05. 06. 07. class Zi extends Fu{ void show(){ 08. 09. System.out.println("zi show"); 10. } 11. 12. 13. class DuoTaiDemo{

public static void main(String[] args){

_ 0 X

Fu f = new Zi();

f.show();

}

画 管理员: C:\Windows\system32\cmd.exe

D:∖code∖day10>java DuoTaiDemo

简单说:编译和运行看左边。

class Zi extends Fu{

}

class Fu{

D:\code\day10>javac DuoTaiDemo.java

编译时:参考的是对象所属的类中是否有调用的函数。

运行时:参考的是对象所属的类中是否有调用的函数。

static void method(){

static void method(){

System.out.println("fu static method");

System.out.println("zi static method");

复制代码

class DuoTaiDemo{ 13. public static void main(String[] args){ 14. Fu f = new Zi(); 15. f.method();// fu static method 16. Fu.method(); 17. 18. } 19. } 复制代码 运行结果: - - X 画 管理员: C:\Windows\system32\cmd.exe D:∖code∖day10>javac DuoTaiDemo.java √day10>java DuoTaiDemo fu static method fu static method 4.9 内部类 定义: 将一个类定义在另一个类的里面,里面那个类就称为内部类(内置类,嵌套类)。

示例1: 01. 内部类的设计: 02. 分析事物时,发现该事物描述中还有事物,而且这个事物还在访问被描述事物的内容,这时候就定义 内部类。 */ 04. class Outer{ 05. private int num = 3; 06. 07. class Inner //内部类 08. 09. 10. void show(){ System.out.println("show run..." + num); 11. 12. }

访问特点: 内部类可以直接访问外部类中的成员,包括私有成员。 而外部类要访问内部类中的成员必须要建立内部类的对象。

public void method(){ 15. Inner in = new Inner(); 16. 17. in.show(); 18. } 19. 20. class InnerClassDemo{ 21. public static void main(String[] args){ 22. Outer out = new Outer(); 23. out.method(); 25. } 26. }

_ 0 X

📷 管理员: C:\Windows\system32\cmd.exe D:\code\day10>javac InnerClassDemo.java D:∖code∖day10>java InnerClassDemo show run...3 示例2: 01. class Outer{ private int num = 3;

02. 03. class Inner 04. 05. void show(){ 06. System.out.println("show run..." + num); 07. 08. } 09. } 10. 11. 12. class InnerClassDemo{

```
//直接访问外部类中的内部类中的成员
 14.
                  Outer.Inner in = new Outer().new Inner();
 15.
 16.
                  in.show();
 17.
             }
 18.
       }
       复制代码
 运行结果:
                                                                    📷 管理员: C:\Windows\system32\cmd.exe
   D:\code\day10>javac InnerClassDemo.java
   D:∖code∖day10>java InnerClassDemo
   show run...3
  内部类的位置:
  内部类定义在成员位置上,可以被private、static成员修饰符修饰。被static修饰的内部类只能访问外部类中
的静态成员。
  示例1:
 01.
       class Outer{
             private static int num = 3;
 02.
 03.
 04.
             static class Inner
 05.
                   void show(){
 06.
 07.
                       System.out.println("show run..." + num);
 08.
                   }
 09.
             }
 10.
 11.
 12.
      class InnerClassDemo{
 13.
             public static void main(String[] args){
                   //如果内部类是静态的,相当于一个外部类
 14.
 15.
                  Outer.Inner in = new Outer.Inner();
                  in.show();
 16.
 17.
             }
      }
 18.
       复制代码
 运行结果:
                                                                    _ O X
   画 管理员: C:\Windows\system32\cmd.exe
```

public static void main(String[] args){

13.

D:\code\day10>javac InnerClassDemo.java D:∖code∖day10>java InnerClassDemo show run...3 示例2:如果内部类是静态的,内部类成员也是静态的,可以不用创建内部类对象,直接调用。

class Outer{ 01. private static int num = 3; 02. 03. static class Inner 04.

static void show(){ System.out.println("show run..." + num); } } class InnerClassDemo{ public static void main(String[] args){ Outer.Inner.show(); }

05. 06. 07. 08. 09. 10. 11. 12. 13. 14. 15. } 16. 复制代码 运行结果: - - X 画 管理员: C:\Windows\system32\cmd.exe D:\code\day10>javac InnerClassDemo.java D:∖code∖day10>java InnerClassDemo show run...3

P.S. 1、如果内部类中定义了静态成员,该内部类也必须是静态的! 示例: class Outer{ 01. private static int num = 3; 02. 03.

static class Inner 04. 05. static void show(){ 06. System.out.println("show run..." + num); 07. } 08. } 09. } 10. 复制代码

2、为什么内部类能直接访问外部类中的成员呢? 那是因为内部类持有了外部类的引用,外部类名.this。 示例: class Outer{ 01. 02. int num = 3; 03. class Inner{ 04. int num = 4; 05. void show(){ int num = 5; 06. System.out.println(num); 07.

System.out.println(this.num); 08. System.out.println(Outer.this.num); 09. 10. } 11. void method(){ 12. new Inner().show(); 13. 14. } 15. 16. class InnerClassDemo{ 17. public static void main(String[] args){ 18. 19. new Outer().method(); 20. 21. } 复制代码 运行结果:

📷 管理员: C:\Windows\system32\cmd.exe D:\code\day10>javac InnerClassDemo.java D:∖code∖day10>java InnerClassDemo 3、内部类定义在局部位置上,也可以直接访问外部类中的成员。 同时可以访问所在局部中的局部变量,但必须是被final修饰的。 示例:

class Outer{ 01. int num = 3;02. void method(final int y){ 04. final int x = 9; 05. class Inner{ void show(){ 06. 07. System.out.println("show..." + x + "," + y); 08. } } 09. 10. Inner in = new Inner(); 11. in.show(); } 12. 13. } 14. 15. class InnerClassDemo{

public static void main(String[] args){ 16. 17. new Outer().method(4); 18. } 19. } 复制代码 运行结果: _ _ _ X 📷 管理员: C:\Windows\system32\cmd.exe D:\code\day10>javac InnerClassDemo.java D:∖code∖day10>java InnerClassDemo show...9,4 匿名内部类

定义: 就是内部类的简化写法。 前提: 内部类可以继承或实现一个外部类或者接口。 格式: new 外部类名或者接口名(){覆盖类或者接口中的代码,(也可以自定义内容。)} 简单理解: 就是建立一个带内容的外部类或者接口的子类匿名对象。

什么时候使用匿名内部类呢? 通常使用方法是接口类型参数,并且该接口中的方法不超过三个,可以将匿名内部类作为参数传递。 好处: 增强阅读性。 示例1: 01. abstract class Demo{ 02. abstract void show(); 03. } 04. 05. class Outer{ int num = 4; 06.

07. 08. void method(){ new Demo(){//匿名内部类 09. void show(){ 10. 11. System.out.println("show....." + num); 12. 13. }.show(); 14. } 15. } 16. 17. class InnerClassDemo{ public static void main(String[] args){ 18. new Outer().method(); 19. 20. } 21. 复制代码 运行结果:

📷 管理员: C:\Windows\system32\cmd.exe

20.

21.

22.

23. 24. 25.

}

运行结果:

复制代码

class InnerClassDemo{

画 管理员: C:\Windows\system32\cmd.exe

D:∖code∖day10>java InnerClassDemo

...show1... ...show2...

对象的初始化过程

示例:

33.

34.

35. 36.

}

void show(){

D:\code\day10>javac InnerClassDemo.java

public static void main(String[] args){

new Outer().method();

_ 0 X

D:\code\day10>javac InnerClassDemo.java D:\code\day10>java InnerClassDemo show.....4 示例2: 01. interface Inter{ void show1(); 02. void show2(); 03. 04. 05. class Outer{ 06. public void method(){ 07. 08. Inter in = new Inter(){ public void show1(){ 09. System.out.println("...show1...."); 10. 11. 12. public void show2(){ 13. System.out.println("...show2...."); } 14. 15. }; 16. in.show1(); in.show2(); 17. 18. } 19.

_ _ _ X 📰 管理员: C:\Windows\system32\cmd.exe D:\code\day10>javac InnerClassDemo.java D:∖code∖day10>java InnerClassDemo ...show1.... ...show2.... 示例3: 01. interface Inter{ 02. void show1(); void show2(); 03. 04. } 05. 06. 通常的使用场景之一: 07. 当函数参数是接口类型时,而且接口中的方法不超过三个。 08. 可以用匿名内部类作为实际参数进行传递。 09. */ 10. class InnerClassDemo{ 11. public static void main(String[] args){ 12. show(new Inter(){ 13. 14. public void show1(){ System.out.println("...show1..."); 15. 16. public void show2(){ 17. System.out.println("...show2..."); 18. 19. } 20. }); 21. public static void show(Inter in){ 22. in.show1(); 23. in.show2(); 24. 25. } 26. } 复制代码 运行结果:

_ 0 X

01. class Fu{ 02. int num = 9;03. 04. { System.out.println("Fu"); 05. } 06. 07. 08. Fu(){ 09. super();//Object //显示初始化 10. //构造代码块初始化 11. 12. show(); 13. } 14. void show(){ System.out.println("fu show " + num);//被覆盖,运行子类的 15. } 16. 17. } 18. 19. class Zi extends Fu{ 20. int num = 8; 21. 22. { System.out.println("Zi"); 23. 24. } 25. 26. Zi(){ 27. super(); //显示初始化 28. //构造代码块初始化 29. 30. show(); 31. } 32.

37. public class Demo{ 38. public static void main(String[] args){ 39. 40. new Zi(); 41. } 42. } 复制代码 运行结果: _ _ _ X 画 管理员: C:\Windows\system32\cmd.exe D:∖code∖day10>javac Demo.java D:∖code∖day10>java Demo zi show 0 zi show 8 ~END~ - Die 40 Die 加大加大加大加大加大加大加大加大加大加大加 ~爱上海,爱黑马~

System.out.println("zi show " + num);