第七章 抽象、接⼝、内部类-作业

**⼀、**abstract**关键****字**

## 基础问答

### 题⽬要求：

请解释abstract关键字的作⽤是什么。

**答：**

用于定义抽象类和抽象方法

## 基础问答

### 题⽬要求：

请简述抽象类的特点。

**答：**

* 1. 必须使用abstract关键词修饰
  2. 相对于普通类，抽象类具有包含抽象方法的能力
  3. 抽象类不能被实例化

## 基础问答

### 题⽬要求：

请回答抽象类及抽象⽅法可以被final修饰符修饰吗?

**答：**

在Java等支持抽象类和抽象方法的编程语言中，abstract 和 final 是互斥的关键字，它们不能同时用于同一个类或方法上。

**抽象类与final关键字**：抽象类是不可以被声明为 final 的。final 关键字意味着类不能被继承，而抽象类存在的意义就是为了让其他类去继承它并实现其中的抽象方法。

**抽象方法与final关键字**：抽象方法也不能被声明为 final。final 方法表示该方法不能被子类覆盖（override）。然而，抽象方法正是为了被子类覆盖而存在的，因为它本身就没有实现，所以也不能声明为 final。

## 程序分析题

### 题⽬要求：

请阅读并分析以下程序，判断以下代码是否正确，如果正确，输出什么信息？如果不正确，请说明原因。

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

abstract class A {

abstract void show();

}

class B extends A {

void show() {

System.out.println("B");

}

}

class Main {

public static void main(String[] args) { A a = new B();

a.show();

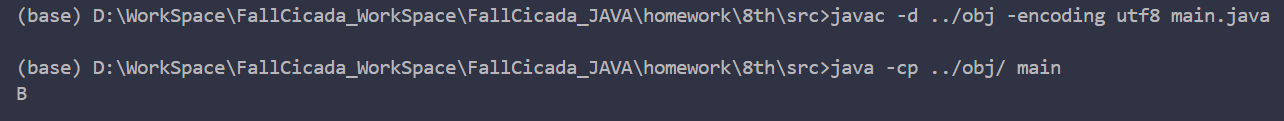
}

}

**答：**代码没错，main类命名应该遵循Java命名规范，首字母大写

**运行：**

1. 抽象类A包含抽象方法show()
2. 声明方法B继承匿名类A重写方法show()，打印：”B”
3. 声明主力main证明主类A a指向B，执行a对象的show()方法打印”B”

**运行结果：  
**

## 编程题

### 题⽬要求：

编写⼀个抽象类 Shape，包含以下属性和⽅法：  属性：颜⾊ color。

 ⽅法：计算形状⾯积的抽象⽅法 area() 和返回颜⾊信息的普通⽅法 getColor()。

然后创建两个具体的⼦类 Rectangle（矩形） 和 Circle（圆形），实现 area() ⽅法并考虑它们各⾃的特点。要求能够使⽤以下测试代码进⾏测试：

|  |  |  |  |
| --- | --- | --- | --- |
| 1  2 | public class Test1\_5\_ShapeTest {  public static void main(String[] args) { | | |
| 3 |  |  | Shape rectangle = new Rectangle("Red", 2.0, 3.0); |
| 4 |  |  | Shape circle = new Circle("Blue", 5); |
| 5 |  |  | System.out.println(rectangle.getColor()); |
| 6 |  |  | System.out.println("Area: " + rectangle.area()); |
| 7 |  |  | System.out.println(); |
| 8 |  |  | System.out.println(circle.getColor()); |
| 9 |  |  | System.out.println("Area: " + circle.area()); |
| 10 |  | } |  |
| 11 | } |  |  |

**答：**

**代码：**

**Shape类**

1. package shape;

2.

3. public abstract class Shape {

4.     private String color;

5.     public Shape() {

6.     }

7.     public Shape(String color) {

8.         this.color = color;

9.     }

10.     public abstract double area();

11.     public String getColor(){

12.         return color;

13.     }

14. }

15.

**Circle类：**

1. package shape;

2. public class Circle extends Shape {

3.     private double radius;

4.

5.     public Circle() {

6.

7.     }

8.     public Circle(String color,double radius) {

9.         super(color);

10.         this.radius = radius;

11.     }

12.     @Override

13.     public double area() {

14.         return Math.PI \* radius \* radius;

15.     }

16.

**Rectangle类：**

1. package shape;

2.

3. public class Rectangle extends Shape {

4.     private double width;

5.     private double height;

6.

7.     public Rectangle() {

8.

9.     }

10.     public Rectangle(String color,double width, double height) {

11.         super(color);

12.         this.width = width;

13.         this.height = height;

14.     }

15.     @Override

16.     public double area() {

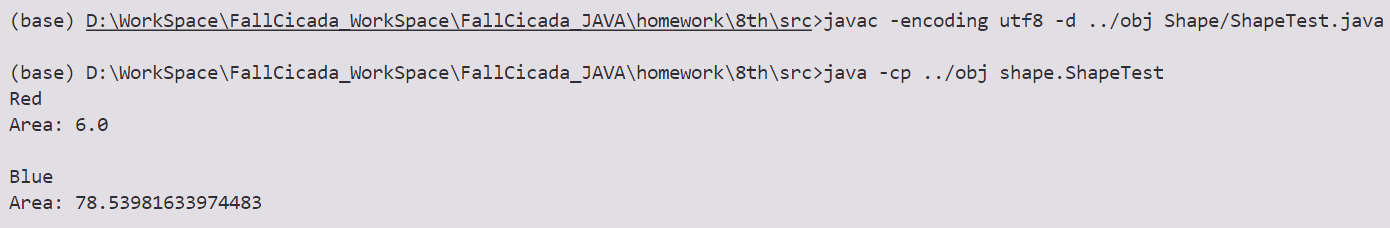
17.         return width \* height;

18.     }

19. }

20.

**运行结果：**

****

## 编程题

### 题⽬要求：

设计⼀个简单的动物园系统，包含以下⼏种动物：猫、狗、⻦。每种动物都有⾃⼰的特点和⾏为，但都有共同的属性和⽅法。其中，猫、狗和⻦都属于动物类，但它们的叫声和⾏为各不相同。

要求：

makeSound()

* 1. 使⽤抽象类来实现动物园系统，包含⼀个抽象⽅法

name

move()

⽤于输出动物的叫声。

* 1. 每种动物都有⼀个属性

表示动物的名称，以及⼀个⽅法

⽤于输出动物的⾏为。

* 1. 猫类具有属性的⾏为。
  2. 狗类具有属性

⾏为。

* 1. ⻦类具有属性

⻦的⾏为。

表示猫的颜⾊，重写

表示狗的年龄，重写

color

makeSound()

age

makeSound()

表示⻦的翼展，重写

wingSpan

⽅法输出猫的叫声，并重写

⽅法输出狗的叫声，并重写

move()

move()

⽅法输出⻦的叫声，并重写

makeSound()

⽅法输出猫

⽅法输出狗的

⽅法输出

move()

要求能够使⽤以下测试代码进⾏测试：

1. class Test1\_6\_Zoo {
2. public static void main(String[] args) {
3. Animal cat = new Cat("tom", "灰⽩");
4. Animal dog = new Dog("旺财", 5);
5. Animal bird = new Bird("樱桃", 21.76);
6. cat.makeSound();//猫发出喵喵叫声
7. cat.move();//猫在四腿爬⾏
8. dog.makeSound();//狗发出汪汪叫声
9. dog.move();//狗在四腿奔跑
10. bird.makeSound();//⻦发出啾啾叫声
11. bird.move();//⻦在⻜翔

12 }

13 }

**答：**

**代码：**

**Animal类：** 1. package zoo;

2.

3. // 抽象类Animal

4. public abstract class Animal {

5. private String name;

6.

7. public Animal(String name) {

8. this.name = name;

9. }

10.

11. public abstract void makeSound();

12.

13. public abstract void move();

14. }

15.

**Bird类：**

1. package zoo;

2.

3. // 鸟类

4. public class Bird extends Animal {

5.     private double wingspan;

6.

7.     public Bird(String name, double wingspan) {

8.         super(name);

9.         this.wingspan = wingspan;

10.     }

11.

12.     @Override

13.     public void makeSound() {

14.         System.out.println("啾啾");

15.     }

16.

17.     @Override

18.     public void move() {

19.         System.out.println("鸟在飞翔");

20.     }

21. }

22.

**Cat类：**

1. package zoo;

2.

3. // 猫类

4. public class Cat extends Animal {

5.     private String color;

6.

7.     public Cat(String name, String color) {

8.         super(name);

9.         this.color = color;

10.     }

11.

12.     @Override

13.     public void makeSound() {

14.         System.out.println("喵喵");

15.     }

16.

17.     @Override

18.     public void move() {

19.         System.out.println("猫在四腿爬行");

20.     }

21. }

22.

**Dog类：**

1. package zoo;

2.

3. // 狗类

4. public class Dog extends Animal {

5.     private int age;

6.

7.     public Dog(String name, int age) {

8.         super(name);

9.         this.age = age;

10.     }

11.

12.     @Override

13.     public void makeSound() {

14.         System.out.println("汪汪");

15.     }

16.

17.     @Override

18.     public void move() {

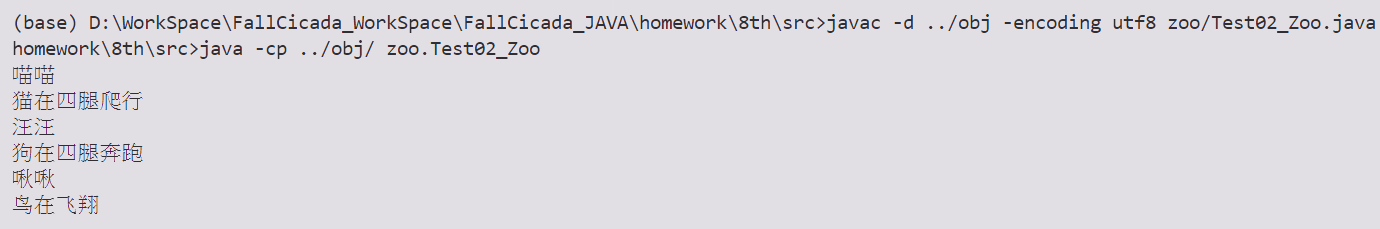
19.         System.out.println("狗在四腿奔跑");

20.     }

21. }

22.

**运行结果：**

****

# ⼆、内部类

## 基础问答

### 题⽬要求：

请回答Java中的内部类有哪⼏种，并说明它们的特点。

**答：**

1. **成员内部类： 成员内部类是作为外部类的一个成员存在的，它可以访问外部类的所有成员变量和方法，包括私有的成员。成员内部类可以是静态的也可以是非静态的。如果它是非静态的，那么它可以通过外部类的一个实例来访问；如果是静态的，则可以直接通过外部类名来访问。**
2. **局部内部类： 局部内部类定义在一个方法或代码块中，只能在那个方法或代码块内使用。局部内部类不能是静态的，因为它是在方法执行期间创建的临时类。局部内部类可以访问外部方法中的最终变量（final variables），即局部变量必须是最终的或者是实际上不可改变的。**
3. **匿名内部类： 匿名内部类没有名字，它通常用于实现一个接口或者继承一个抽象类，主要用于简化代码，使得可以在声明对象的同时就给它赋予具体的实现。匿名内部类通常用在只需要一次性的使用某个类的地方。**
4. **静态嵌套类： 静态嵌套类也是定义在另一个类中的类，但是它不是内部类的一个成员，因此它不需要访问外部类的实例就可以创建。静态嵌套类前需要加上关键字static，并且它可以通过外部类的名字直接访问，不需要实例化外部类。**

## 编程题

### 题⽬要求：

按照要求，补⻬代码，要求在控制台输出 HelloWorld。

|  |  |  |
| --- | --- | --- |
| 1 | interface Inner { |  |
| 2 | void show(); |
| 3 | } |
| 4 |  |
| 5 | class Outer { |
| 6 | //这⾥补全代码 提示：匿名内部类 |
| 7 | } |
| 8 |  |
| 9 | class Test2\_2\_InnerClass{ |
| 10 | public static void main(String[] args) | { |
| 11 | Outer.method().show(); |  |
| 12 | } |  |
| 13 | } |  |

**答：**

**代码：**class Outer {

    // 使用内部类实现 show 方法

    public static class Inner {

        public *void* show() {

            System.out.println("HelloWorld");

        }

    }

    // 返回内部类的实例

    public static Inner method() {

        return **new** Inner();

    }

}

class Test2\_2\_InnerClass {

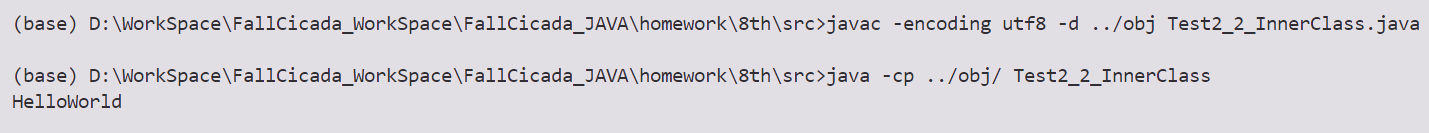
    public static *void* main(String[] *args*) {

        Outer.method().show();

    }

}

**运行结果：**



## 编程题

### 题⽬要求：

使⽤匿名内部类⽅式实现下⾯接⼝，并完成规定的功能。

1. interface Action{
2. public String[] test(String str);

3 }

对该接⼝进⾏实现，让其可以具有处理以下两种字符串的功能：

* 1. 对字符串按照 | 进⾏分割，并得到数组，例如 传⼊字符串"a|b|c"，返回["a","b","c"]
  2. 对字符串按照 - 进⾏分割，并得到数组，例如 传⼊字符串"a-b-c"，返回["a","b","c"]要求使⽤匿名内部类的⽅式来完成。

提示：**分割字符串，学习**String**类中**split**⽅法实现。**

 interface **Action** {

**String**[] **test**(**String** str);

 }

 public class **Test2\_3** {

     public static void **main**(**String**[] args) {

*// 使用匿名内部类实现按 "|" 分割字符串的功能*

**Action** actionPipe = new **Action**() {

             @**Override**

             public **String**[] **test**(**String** str) {

                 return str.**split**("\\|"); *// 使用 "|" 作为分隔符*

             }

         };

*// 使用匿名内部类实现按 "-" 分割字符串的功能*

**Action** actionDash = new **Action**() {

             @**Override**

             public **String**[] **test**(**String** str) {

                 return str.**split**("-"); *// 使用 "-" 作为分隔符*

             }

         };

*// 测试按 "|" 分割的方法*

**String**[] pipeSplit = actionPipe.**test**("a|b|c");

*// 构造并打印结果*

**System**.out.**println**(**formatArray**(pipeSplit)); *// 输出 "[\"a\",\"b\",\"c\"]"*

*// 测试按 "-" 分割的方法*

**String**[] dashSplit = actionDash.**test**("a-b-c");

*// 构造并打印结果*

**System**.out.**println**(**formatArray**(dashSplit)); *// 输出 "[\"a\",\"b\",\"c\"]"*

     }

*// 辅助方法用于格式化字符串数组*

     private static **String** **formatArray**(**String**[] array) {

**StringBuilder** sb = new **StringBuilder**("[\"");

         for (int i = 0; i < array.length; i++) {

             sb.**append**(array[i]);

             if (i < array.length - 1) {

                 sb.**append**("\",\"");

             }

         }

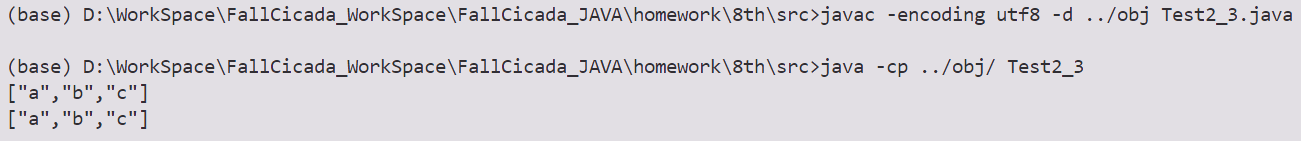
         sb.**append**("\"]");

         return sb.**toString**();

     }

 }

**运行结果：**



# 三、访问控制

## 基础问答

### 题⽬要求：

请描述各个权限修饰符的作⽤。

**答：**

1. **public:**
   * **public 是最开放的权限修饰符，表明声明为 public 的成员在整个应用程序中都是可见的。**
   * **public 成员可以在任何地方被访问，包括不同的包和类。**
2. **protected:**
   * **protected 表明声明为 protected 的成员可以在同一个包内或者子类中访问。**
   * **如果子类不在同一个包内，那么它依然可以通过继承的方式访问 protected 成员。**
3. **默认权限（无修饰符）:**
   * **当没有显式指定任何权限修饰符时，默认权限适用于类的成员。**
   * **默认权限意味着成员仅限于同一个包内的类访问。**
4. **private:**
   * **private 是最严格的权限修饰符，表明声明为 private 的成员只能在其声明的类内部访问。**
   * **private 成员不能被子类访问，也不能在同一个包外访问。**

**四、**interface

## 基础问答

### 题⽬要求：

请简述什么情况下该⽤抽象类、什么情况下该⽤接⼝。

**答：**

**抽象类的使用场景：**

1. **共享实现**：当多个类具有相似的行为并且希望共享这部分行为的实现时，可以考虑使用抽象类。抽象类可以提供部分实现，子类可以选择继承并重写这些方法。
2. **模板方法模式**：抽象类非常适合用于模板方法模式中，其中抽象类定义了算法的骨架，而将某些步骤延迟到子类去实现。
3. **提供默认行为**：抽象类可以提供一些默认的方法实现，子类可以在必要时覆盖这些方法。
4. **避免子类随意更改父类的实现**：通过声明方法为abstract，强制子类去实现这些方法，从而确保所有子类都遵循一定的规范。

**接口的使用场景：**

1. **完全抽象**：接口是完全抽象的，它只定义方法签名而不提供任何具体的实现。当需要定义一个行为的标准但不需要关心具体实现时，使用接口。
2. **多重继承**：Java不支持类的多重继承，但是一个类可以实现多个接口，因此在需要实现多个抽象类型的行为时，使用接口是一个很好的选择。
3. **松耦合系统**：接口提供了良好的松耦合机制，使得代码更加灵活，更容易进行单元测试。
4. **定义标准**：接口常常用来定义行业标准或框架约定，比如Java集合框架中的List, Map等都是由接口定义的。

## 编程题

### 题⽬要求：

根据以下描述编写程序。

⻦类（Bird）可以⻜翔，⻜机类（Plane）也可以⻜翔，那么可以把⻜翔（flyAction）这个动作抽象出来，抽象到

⼀个抽象类或者接⼝中，然后让⻦类和⻜机类去继承或实现，那么这⾥是把⽅法抽象到⽗类型中合适，还是抽象到接⼝中合适？编写代码完成本题。

要求能够使⽤以下测试代码进⾏测试:

1. class Test4\_2\_Interface {
2. public static void main(String[] args) {
3. Fly bird = new Bird();
4. bird.flyAction(); //⻦会⻜⾏
5. Fly plane = new Plane();
6. plane.flyAction(); //⻜机能⻜⾏

7 }

8 }

答：

代码：

IFly接口：

package **flight**;

public interface **IFly** {

    void **flyAction**();

}

Bird类：

package **flight**;

class **Bird** implements **IFly** {

    @**Override**

    public void **flyAction**() {

**System**.out.**println**("鸟能飞翔");

    }

}

Plane类：

package **flight**;

public class **Plane** implements **IFly** {

    @**Override**

    public void **flyAction**() {

**System**.out.**println**("飞机能飞翔");

    }

}

测试类：

package **flight**;

public class **Test4\_2\_Interface** {

    public static void **main**(**String**[] args) {

**IFly** bird = new **Bird**();

        bird.**flyAction**(); *// 鸟会飞翔*

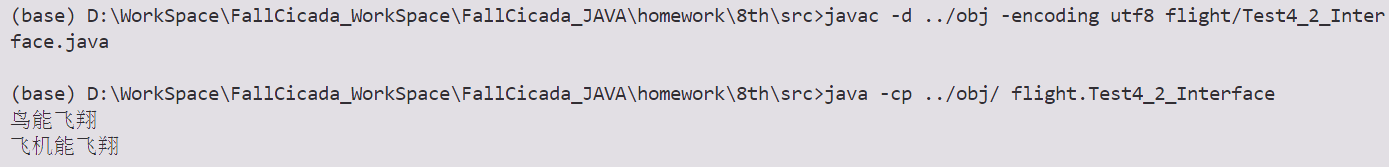
**IFly** plane = new **Plane**();

        plane.**flyAction**(); *// 飞机能飞翔*

    }

}

**运行结果：**

****

## 编程题

### 题⽬要求：

按照以下要求完成编程题。

1

2

3

4

5

6

7

8

9

10

11

定义⼀个Person类

属性：name age

⾏为：void run();

再定义⼀个Student类

属性：id score

⾏为：void study();

给Student类增强功能功能1：唱歌

功能2：⻜⾏

接⼝ ISingAble { void sing();}

接⼝ IFlyAble{ void fly(); }

答：

代码：

IFlyAble接口：

package **student**;

*// 定义 IFlyAble 接口*

public interface **IFlyAble** {

    void **fly**();

}

ISingAble接口：

package **student**;

*// 定义 ISingAble 接口*

public interface **ISingAble** {

    void **sing**();

}*// 定义 ISingAble 接口*

Person类：

package **student**;

*// 定义 Person 类*

public class **Person** {

**String** name;

    int age;

*// 构造方法*

    public **Person**(**String** name, int age) {

        this.name = name;

        this.age = age;

    }

*// 行为：run*

    public void **run**() {

**System**.out.**println**(name + " is running.");

    }

}

Student类:

package **student**;

*// 定义 Student 类继承自 Person 类*

public class **Student** extends **Person** implements **ISingAble**, **IFlyAble** {

**String** id;

    double score;

*// 构造方法*

    public **Student**(**String** name, int age, **String** id, double score) {

        super(name, age);  *// 调用父类构造器*

        this.id = id;

        this.score = score;

    }

*// 行为：study*

    public void **study**() {

**System**.out.**println**(super.name + " is studying hard.");

    }

*// 实现 ISingAble 接口中定义的方法*

    @**Override**

    public void **sing**() {

**System**.out.**println**(super.name + " is singing a song.");

    }

*// 实现 IFlyAble 接口中定义的方法*

    @**Override**

    public void **fly**() {

**System**.out.**println**(super.name + " is flying.");

    }

}

测试类:

package **student**;

*// 测试代码*

public class **Test4\_3** {

    public static void **main**(**String**[] args) {

**Student** student = new **Student**("Tom", 20, "123456", 90.5);

        student.**run**();    *// 继承自 Person 的行为*

        student.**study**();  *// Student 自身的行为*

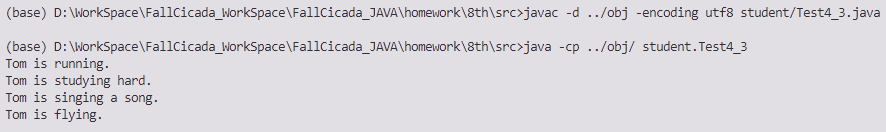
        student.**sing**();   *// 实现 ISingAble 接口中的行为*

        student.**fly**();    *// 实现 IFlyAble 接口中的行为*

    }

}

运行结果：



# 五、包装类

## 基础问答

### 题⽬要求：

请写出基本数据类型对应的的包装类名。

**答：**

* **int 的包装类为 Integer**
* **byte 的包装类为 Byte**
* **short 的包装类为 Short**
* **long 的包装类为 Long**
* **float 的包装类为 Float**
* **double 的包装类为 Double**
* **char 的包装类为 Character**
* **boolean 的包装类为 Boolean**

## 程序分析题

### 题⽬要求：

请阅读并分析以下程序案例，写出程序运⾏输出的结果。

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

Integer i1 = new Integer(97);

Integer i2 = new Integer(97); System.out.println(i1 == i2); System.out.println(i1.equals(i2));

Integer i3 = new Integer(148);

Integer i4 = new Integer(148); System.out.println(i3 == i4); System.out.println(i3.equals(i4));

Integer i5 = 97; Integer i6 = 97;

System.out.println(i5 == i6); System.out.println(i5.equals(i6));

Integer i7 = 148; Integer i8 = 148;

18

19

20

21

22

23

24

25

System.out.println(i7 == i8);

System.out.println(i7.equals(i8));

int a = 148;

int b = 97; Integer c = 97;

System.out.println(i7 == a);

System.out.println(b == c);

**代码及注释：**

*// 创建两个值相同的Integer对象，使用new关键字*

**Integer** i1 = new **Integer**(97);

**Integer** i2 = new **Integer**(97);

*// 输出两个对象的引用是否相同，false*

**System**.out.**println**(i1 == i2);

*// 使用equals方法比较两个对象的值，true*

**System**.out.**println**(i1.**equals**(i2));

*// 创建两个值相同的Integer对象，使用new关键字，值为148*

**Integer** i3 = new **Integer**(148);

**Integer** i4 = new **Integer**(148);

*// 输出两个对象的引用是否相同，false*

**System**.out.**println**(i3 == i4);

*// 使用equals方法比较两个对象的值，true*

**System**.out.**println**(i3.**equals**(i4));

*// 使用Integer缓存池中的对象，值为97*

**Integer** i5 = 97;

**Integer** i6 = 97;

*// 输出两个对象的引用是否相同，true，因为值在缓存范围内*

**System**.out.**println**(i5 == i6);

*// 使用equals方法比较两个对象的值，true*

**System**.out.**println**(i5.**equals**(i6));

*// 使用Integer缓存池中的对象，值为148*

**Integer** i7 = 148;

**Integer** i8 = 148;

*// 输出两个对象的引用是否相同，true，因为值在缓存范围内*

**System**.out.**println**(i7 == i8);

*// 使用equals方法比较两个对象的值，true*

**System**.out.**println**(i7.**equals**(i8));

*// 定义基本类型int变量a，值为148*

        int a = 148;

*// 定义基本类型int变量b，值为97*

        int b = 97;

*// 定义Integer对象c，值为97，从缓存中获取*

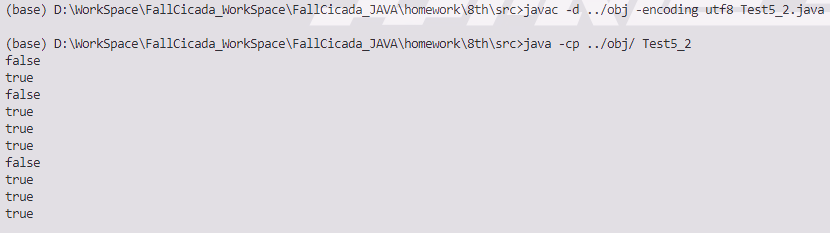
**Integer** c = 97;

*// 比较Integer对象i7和基本类型int变量a的值，true*

**System**.out.**println**(i7 == a);

*// 比较基本类型int变量b和Integer对象c的值，true*

**System**.out.**println**(b == c);

**运行结果：  
**

**六、**Object**类**

## 基础问答

### 题⽬要求：

请简述==和equals()的区别。

**答：**

* **== 用于比较两个变量或对象引用是否指向同一个内存地址，即比较的是对象的引用。**
* **equals() 方法则用于比较两个对象的内容是否相等，即比较的是对象的值。**

## 程序分析题

### 题⽬要求：

请阅读并分析下⾯程序的运⾏结果。

1

2

3

4

5

6

7

8

9

10

11

12

13

class A {

}

class B extends A {

void test() {

System.out.println(super.getClass().getSimpleName()); System.out.println(this.getClass().getSimpleName());

}

public static void main(String[] args) {

new B().test();

}

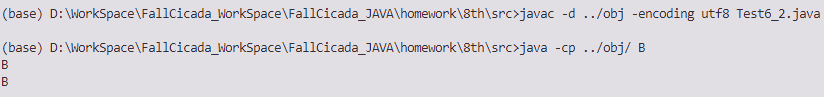
}

**答：**

* **super.getClass().getSimpleName() 返回的是当前对象的类名称，即 B。**
* **this.getClass().getSimpleName() 同样返回当前对象的类名称，即 B。**

**在 Java 中，super 关键字通常用于访问父类的方法或属性。然而，super.getClass() 实际上返回的是当前对象的类，而不是父类的类。因此，无论 super 关键字指向哪个父类，super.getClass() 都会返回当前对象的类。**

**运行结果：**

****

**七、**String**字符串⽐较问****题**

## 程序分析

### 题⽬要求：

阅读并分析以下两个案例中代码的运⾏结果。

**案例⼀：**

1. String str1 = "hello";
2. String str2 = "hello";
3. String str3 = new String("hello");
4. String str4 = new String("hello");
5. String str5 = "hellohello";
6. String str6 = str1 + str2;
7. System.out.println(str1 == str2);
8. System.out.println(str2 == str3);
9. System.out.println(str3 == str4);
10. System.out.println(str2.equals(str3));
11. System.out.println(str3.equals(str4));
12. System.out.println(str5.equals(str6));
13. System.out.println(str5 == str6);

**答：**

**代码及思路注释：**

public class TestString\_01 {

    public static *void* main(String[] *args*) {

        // 字符串字面量，它们在字符串常量池中存储

        String str1 = "hello";

        String str2 = "hello";

        // 通过new关键字创建的字符串对象，它们在堆内存中存储

        String str3 = **new** String("hello");

        String str4 = **new** String("hello");

        // 字符串拼接操作，结果存储在另一个字符串变量中

        String str5 = "hellohello";

        String str6 = str1 + str2;

        // 使用==比较字符串对象的引用，展示str1和str2引用相同对象

        System.out.println(str1 == str2);// true

        // 展示str2和str3虽然内容相同，但引用不同

        System.out.println(str2 == str3);// false

        // 展示str3和str4都是通过new创建的，所以引用不同

        System.out.println(str3 == str4);// false

        // 使用equals方法比较字符串的内容，展示str2和str3内容相同

        System.out.println(str2.equals(str3));// true

        // 再次验证通过equals方法，str3和str4内容相同

        System.out.println(str3.equals(str4));// true

        // 验证str5和str6内容相同但不是同一个对象引用

        System.out.println(str5.equals(str6));// true

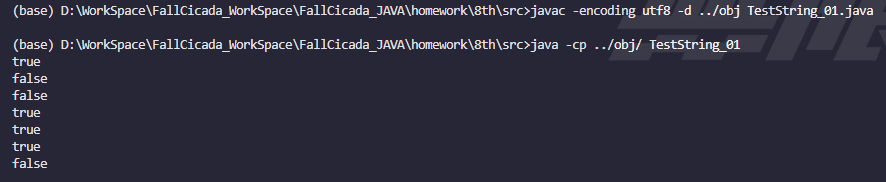
        // 验证str5和str6虽然内容相同，但使用==比较时为false，因为它们是不同的对象引用

        System.out.println(str5 == str6);// false

    }

}

**运行结果：**

****

**案例⼆：**

1. String s1 = "helloworld";
2. String s2 = "hello";
3. String s3 = "world";
4. String s4 = "hello" + "world";
5. String s5 = s2 + s3;
6. String s6 = new String("helloword");
7. String s7 = "hello" + new String("world");
8. System.out.println(s1 == s4);
9. System.out.println(s1 == s5);
10. System.out.println(s1 == s6);
11. System.out.println(s1 == s7);

**答：**

**代码及其思路注释：**

public class TestString\_02{

    public static *void* main(String[] *args*){

        // 初始化一个字符串常量，表示"helloworld"

        String s1 = "helloworld";

        // 初始化两个字符串常量，分别表示"hello"和"world"

        String s2 = "hello";

        String s3 = "world";

        // 通过直接连接两个字符串常量创建一个新的字符串常量，表示"helloworld"

        String s4 = "hello" + "world";

        // 通过连接两个字符串变量创建一个新的字符串常量，表示"helloworld"

        String s5 = s2 + s3;

        // 初始化一个新的字符串对象，表示"helloword"

        String s6 = **new** String("helloword");

        // 通过连接一个字符串常量和一个新的字符串对象创建一个新的字符串对象，表示"helloworld"

        String s7 = "hello" + **new** String("world");

        // 比较s1和s4是否指向相同的字符串常量池对象，输出true，因为它们是相同的常量

        System.out.println(s1 == s4);

        // 比较s1和s5是否指向相同的字符串对象，输出false，因为字符串拼接会在堆上创建新对象

        System.out.println(s1 == s5);

        // 比较s1和s6是否指向相同的字符串对象，输出false，因为s6是一个不同的对象

        System.out.println(s1 == s6);

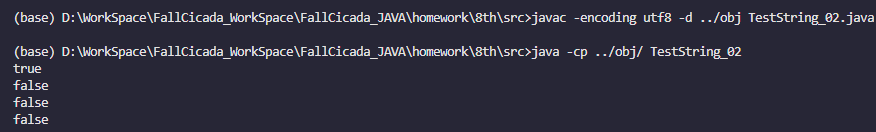
        // 比较s1和s7是否指向相同的字符串对象，输出false，因为字符串拼接会导致新对象的创建

        System.out.println(s1 == s7);

    }

}

**运行结果：**

****

## 基础问答

### 题⽬要求：

请回答String类型与基本数据类型与包装类如何转换？