day03【final、static、接口】

今日内容

- final关键字
- static
- 接口

教学目标

- [ ] 描述final修饰的类的特点
- [ ] 描述final修饰的方法的特点
- [ ] 描述final修饰的变量的特点
- [ ] 能够掌握static关键字修饰的变量调用方式
- [ ] 能够掌握static关键字修饰的方法调用方式
- [ ] 能够写出接口的定义格式
- [ ] 能够写出接口的实现格式
- [ ] 能够说出接口中的成员特点

第一章 final关键字

1.1 概述

学习了继承后，我们知道，子类可以在父类的基础上改写父类内容，比如，方法重写。那么我们能不能随意的继承API中提供的类，改写其内容呢？显然这是不合适的。为了避免这种随意改写的情况，Java提供了`final`关键字，用于修饰**不可改变**内容。

- **final**： 不可改变。可以用于修饰类、方法和变量。
  - 类：被修饰的类，不能被继承。
  - 方法：被修饰的方法，不能被重写。
  - 变量：被修饰的变量，不能被重新赋值。

1.2 使用方式

修饰类

格式如下：

```
final class 类名 {

}
```

查询API发现像 `public final class String` 、`public final class Math` 、`public final class Scanner` 等，很多我们学习过的类，都是被final修饰的，目的就是供我们使用，而不让我们所以改变其内容。

修饰方法

格式如下：

```
修饰符 final 返回值类型 方法名(参数列表){
    //方法体
}
```

重写被 `final` 修饰的方法，编译时就会报错。

修饰变量

- **局部变量——基本类型**

基本类型的局部变量，被final修饰后，只能赋值一次，不能再更改。代码如下：

```
public class FinalDemo1 {
    public static void main(String[] args) {
        // 声明变量，使用final修饰
        final int a;
        // 第一次赋值
        a = 10;
        // 第二次赋值
        a = 20; // 报错,不可重新赋值
    }
}
```

思考，如下两种写法，哪种可以通过编译?

写法1：

```
final int c = 0;
for (int i = 0; i < 10; i++) {
    c = i;
    System.out.println(c);
}
```

写法2：

```
for (int i = 0; i < 10; i++) {
    final int c = i;
    System.out.println(c);
}
```

根据 `final` 的定义，写法1报错！写法2，为什么通过编译呢？因为每次循环，都是一次新的变量c。这也是大家需要注意的地方。

- **局部变量——引用类型**

引用类型的局部变量，被final修饰后，只能指向一个对象，地址不能再更改。但是不影响对象内部的成员变量值的修改，代码如下：

```
public class FinalDemo2 {
    public static void main(String[] args) {
        // 创建 User 对象
        final   User u = new User();

        // 创建 另一个 User对象
        // u = new User(); // 报错，指向了新的对象，地址值改变。

        // 调用setName方法
        u.setName("张三"); // 可以修改
    }
}
```

- **成员变量**

成员变量涉及到初始化的问题，初始化方式有两种，只能二选一：

1. 显示初始化；

```
public class User {
    final String USERNAME = "张三";
    private int age;
}
```

2. 构造方法初始化。

```
public class User {
    final String USERNAME ;
    private int age;
    public User(String username, int age) {
        this.USERNAME = username;
        this.age = age;
    }
}
```

> 被final修饰的常量名称，一般都有书写规范，所有字母都**大写**。

> 小结：final修饰类，类不能被继承。 final修饰方法，方法不能被重写。final修饰变量，变量不能被改值。

第二章 static关键字

2.1 概述

static是静态修饰符，一般修饰成员。被static修饰的成员属于类，不属于单个这个类的某个对象。static修饰的成员被多个对象共享。static修饰的成员属于类，但是会影响每一个对象。被static修饰的成员又叫类成员，不叫对象的成员。

2.2 定义和使用格式

类变量

当 `static` 修饰成员变量时，该变量称为**类变量**。该类的每个对象都**共享**同一个类变量的值。任何对象都可以更改该类变量的值，但也可以在不创建该类的对象的情况下对类变量进行操作。

- **类变量**：使用 static关键字修饰的成员变量。

定义格式：

```
static 数据类型 变量名;
```

举例：

```
static String room;
```

比如说，同学们来黑马程序员学校学习,那么我们所有同学的学校都是黑马程序员, 不因每个同学不同而不同。

所以，我们可以这样定义一个静态变量school，代码如下：

```
public class Student {
  private String name;
  private int age;
  // 类变量，记录学生学习的学校
  public static String school = "黑马程序员学校";

  public Student(String name, int age){
    this.name = name;
    this.age = age;
  }

  // 打印属性值
  public void show() {
    System.out.println("name=" + name + ", age=" + age + ", shool=" + shool );
  }
}

public class StuDemo {
  public static void main(String[] args) {
    Student s1 = new Student("张三", 23);
    Student s2 = new Student("李四", 24);
    Student s3 = new Student("王五", 25);
    Student s4 = new Student("赵六", 26);

    s1.show(); // Student : name=张三, age=23, shool=黑马程序员学校
    s2.show(); // Student : name=李四, age=24, shool=黑马程序员学校
    s3.show(); // Student : name=王五, age=25, shool=黑马程序员学校
    s4.show(); // Student : name=赵六, age=26, shool=黑马程序员学校
  }
}
```

静态方法

当`static`修饰成员方法时，该方法称为**类方法** 。静态方法在声明中有`static`，建议使用类名来调用，而不需要创建类的对象。调用方式非常简单。

- **类方法**：使用 static关键字修饰的成员方法，习惯称为**静态方法。**

定义格式：

```
修饰符 static 返回值类型 方法名 (参数列表){
    // 执行语句
}
```

举例：在Student类中定义静态方法

```
public static void showNum() {
  System.out.println("num:" +  numberOfStudent);
}
```

- **静态方法调用的注意事项：**
  - 静态方法可以直接访问类变量和静态方法。
  - 静态方法**不能直接访问**普通成员变量或成员方法。成员方法可以直接访问类变量或静态方法。
  - 静态方法中，不能使用**this**关键字。

> 小贴士：静态方法只能访问静态成员。

调用格式

被static修饰的成员可以并且建议通过**类名直接访问。**虽然也可以通过对象名访问静态成员，原因即多个对象均属于一个类，共享使用同一个静态成员，但是不建议，会出现警告信息。

格式：

```
// 访问类变量
类名.类变量名;

// 调用静态方法
类名.静态方法名(参数);
```

调用演示，代码如下：

```
public class StuDemo2 {
  public static void main(String[] args) {
    // 访问类变量
    System.out.println(Student.numberOfStudent);
    // 调用静态方法
    Student.showNum();
  }
}
```

> 小结：static修饰的内容是属于类的，可以通过类名直接访问

第三章 接口

3.1 概述

接口，是Java语言中一种引用类型，是方法的集合，如果说类的内部封装了成员变量、构造方法和成员方法，那么接口的内部主要就是**封装了方法**，包含抽象方法（JDK 7及以前），默认方法和静态方法（JDK 8）。

接口的定义，它与定义类方式相似，但是使用 `interface` 关键字。它也会被编译成.class文件，但一定要明确它并不是类，而是另外一种引用数据类型。

public class 类名.java-->.class

public interface 接口名.java-->.class

> 引用数据类型：数组，类，接口。

接口的使用，它不能创建对象，但是可以被实现（`implements`，类似于被继承）。一个实现接口的类（可以看做是接口的子类），需要实现接口中所有的抽象方法，创建该类对象，就可以调用方法了，否则它必须是一个抽象类。

3.2 定义格式

```
public interface 接口名称 {
    // 抽象方法
    // 默认方法
    // 静态方法
}
```

含有抽象方法

抽象方法：使用 `abstract` 关键字修饰，可以省略，没有方法体。该方法供子类实现使用。

代码如下：

```
public interface InterFaceName {
    public abstract void method();
}
```

含有默认方法和静态方法

默认方法：使用 `default` 修饰，不可省略，供子类调用或者子类重写。

静态方法：使用 `static` 修饰，供接口直接调用。

代码如下：

```
public interface InterFaceName {
    public default void method() {
        // 执行语句
    }
    public static void method2() {
        // 执行语句
    }
}
```

> 小结：定义接口时就是将定义类的class改成了interface，并且接口中的内容也有了一些变化。

3.3 基本的实现

实现的概述

类与接口的关系为实现关系，即**类实现接口**，该类可以称为接口的实现类，也可以称为接口的子类。实现的动作类似继承，格式相仿，只是关键字不同，实现使用 `implements` 关键字。

非抽象子类实现接口：

1. 必须重写接口中所有抽象方法。
2. 继承了接口的默认方法，即可以直接调用，也可以重写。

实现格式：

```
class 类名 implements 接口名 {
    // 重写接口中抽象方法【必须】
    // 重写接口中默认方法【可选】
}
```

抽象方法的使用

必须全部实现，代码如下：

定义接口：

```
public interface LiveAble {
    // 定义抽象方法
    public abstract void eat();
    public abstract void sleep();
}
```

定义实现类：

```
public class Animal implements LiveAble {
    @Override
    public void eat() {
        System.out.println("吃东西");
    }

    @Override
    public void sleep() {
        System.out.println("晚上睡");
    }
}
```

定义测试类：

```
public class InterfaceDemo {
    public static void main(String[] args) {
        // 创建子类对象
        Animal a = new Animal();
        // 调用实现后的方法
        a.eat();
        a.sleep();
    }
}
输出结果：
吃东西
晚上睡
```

默认方法的使用

可以继承，可以重写，二选一，但是只能通过实现类的对象来调用。

1. 继承默认方法，代码如下：

定义接口：

```
public interface LiveAble {
    public default void fly(){
        System.out.println("天上飞");
    }
}
```

定义实现类：

```
public class Animal implements LiveAble {
    // 继承，什么都不用写，直接调用
}
```

定义测试类：

```
public class InterfaceDemo {
    public static void main(String[] args) {
        // 创建子类对象
        Animal a = new Animal();
        // 调用默认方法
        a.fly();
    }
}
输出结果：
天上飞
```

1. 重写默认方法，代码如下：

定义接口：

```
public interface LiveAble {
    public default void fly(){
        System.out.println("天上飞");
    }
}
```

定义实现类：

```
public class Animal implements LiveAble {
    @Override
    public void fly() {
        System.out.println("自由自在的飞");
    }
}
```

定义测试类：

```
public class InterfaceDemo {
    public static void main(String[] args) {
        // 创建子类对象
        Animal a = new Animal();
        // 调用重写方法
        a.fly();
    }
}
输出结果：
自由自在的飞
```

静态方法的使用

静态与.class 文件相关，只能使用接口名调用，不可以通过实现类的类名或者实现类的对象调用，代码如下：

定义接口：

```
public interface LiveAble {
    public static void run(){
        System.out.println("跑起来~~~");
    }
}
```

定义实现类：

```
public class Animal implements LiveAble {
    // 无法重写静态方法
}
```

定义测试类：

```
public class InterfaceDemo {
    public static void main(String[] args) {
        // Animal.run(); // 【错误】无法继承方法，也无法调用
        LiveAble.run(); //
    }
}
输出结果：
跑起来~~~
```

> 小结： 类实现接口使用的是implements关键字，并且一个普通类实现接口，必须要重写接口中的所有的抽象方法

3.4 接口的多实现

之前学过，在继承体系中，一个类只能继承一个父类。而对于接口而言，一个类是可以实现多个接口的，这叫做接口的**多实现**。并且，一个类能继承一个父类，同时实现多个接口。

实现格式：

```
class 类名 [extends 父类名] implements 接口名1,接口名2,接口名3... {
    // 重写接口中抽象方法【必须】
    // 重写接口中默认方法【不重名时可选】
}
```

> []：表示可选操作。

抽象方法

接口中，有多个抽象方法时，实现类必须重写所有抽象方法。**如果抽象方法有重名的，只需要重写一次。**代码如下：

定义多个接口：

```
interface A {
    public abstract void showA();
    public abstract void show();
}

interface B {
    public abstract void showB();
    public abstract void show();
}
```

定义实现类：

```
public class C implements A,B{
    @Override
    public void showA() {
        System.out.println("showA");
    }

    @Override
    public void showB() {
        System.out.println("showB");
    }

    @Override
    public void show() {
        System.out.println("show");
    }
}
```

默认方法

接口中，有多个默认方法时，实现类都可继承使用。**如果默认方法有重名的，必须重写一次。**代码如下：

定义多个接口：

```
interface A {
    public default void methodA(){}
    public default void method(){}
}

interface B {
    public default void methodB(){}
    public default void method(){}
}
```

定义实现类：

```
public class C implements A,B{
    @Override
    public void method() {
        System.out.println("method");
    }
}
```

静态方法

接口中，存在同名的静态方法并不会冲突，原因是只能通过各自接口名访问静态方法。

优先级的问题

当一个类，既继承一个父类，又实现若干个接口时，父类中的成员方法与接口中的默认方法重名，子类就近选择执行父类的成员方法。代码如下：

定义接口：

```
interface A {
    public default void methodA(){
        System.out.println("AAAAAAAAAAA");
    }
}
```

定义父类：

```
class D {
    public void methodA(){
        System.out.println("DDDDDDDDDDD");
    }
}
```

定义子类：

```
class C extends D implements A {
    // 未重写methodA方法
}
```

定义测试类：

```
public class Test {
    public static void main(String[] args) {
        C c = new C();
        c.methodA();
    }
}
输出结果：
DDDDDDDDDDD
```

> 小结： 一个类可以实现多个接口，多个接口之间使用逗号隔开即可。

3.5 接口的多继承【了解】

一个接口能继承另一个或者多个接口，这和类之间的继承比较相似。接口的继承使用 `extends` 关键字，子接口继承父接口的方法。**如果父接口中的默认方法有重名的，那么子接口需要重写一次。**代码如下：

定义父接口：

```
interface A {
    public default void method(){
        System.out.println("AAAAAAAAAAAAAAAAAA");
    }
}

interface B {
    public default void method(){
        System.out.println("BBBBBBBBBBBBBBBBBB");
    }
}
```

定义子接口：

```
interface D extends A,B{
    @Override
    public default void method() {
        System.out.println("DDDDDDDDDDDDD");
    }
}
```

> 小贴士：
>
> 子接口重写默认方法时，default关键字可以保留。
>
> 子类重写默认方法时，default关键字不可以保留。

> 小结：接口和接口之间是继承的关系，而不是实现。一个接口可以继承多个接口。

3.6 其他成员特点

- 接口中，无法定义成员变量，但是可以定义常量，其值不可以改变，默认使用public static final修饰。
- 接口中，没有构造方法，不能创建对象。
- 接口中，没有静态代码块。

3.7 抽象类和接口的练习

通过实例进行分析和代码演示抽象类和接口的用法。

1、举例：

犬：

行为：

吼叫；

吃饭；

缉毒犬：

行为：

吼叫；

吃饭；

缉毒；

2、思考：

由于犬分为很多种类，他们吼叫和吃饭的方式不一样，在描述的时候不能具体化，也就是吼叫和吃饭的行为不能明确。当描述行为时，行为的具体动作不能明确，这时，可以将这个行为写为抽象行为，那么这个类也就是抽象类。

可是有的犬还有其他额外功能，而这个功能并不在这个事物的体系中，例如：缉毒犬。缉毒的这个功能有好多种动物都有，例如：缉毒猪，缉毒鼠。我们可以将这个额外功能定义接口中，让缉毒犬继承犬且实现缉毒接口，这样缉毒犬既具备犬科自身特点也有缉毒功能。

```
//定义缉毒接口 缉毒的词组(anti-Narcotics)比较长,在此使用拼音替代
interface JiDu{
    //缉毒
    public abstract void jiDu();
}
//定义犬科,存放共性功能
abstract class Dog{
    //吃饭
    public abstract void eat();
    //吼叫
    public abstract void roar();
}
//缉毒犬属于犬科一种，让其继承犬科，获取的犬科的特性，
//由于缉毒犬具有缉毒功能，那么它只要实现缉毒接口即可，这样即保证缉毒犬具备犬科的特性，也拥有了缉毒的功能
class JiDuQuan extends Dog implements JiDu{
    public void jiDu() {
    }
    void eat() {
```

```
    }
    void roar() {
    }
}

//缉毒猪
class JiDuZhu implements JiDu{
    public void jiDu() {
    }
}
```

> 讲完抽象类和接口后,相信有许多同学会存有疑惑,两者的共性那么多,只留其中一种不就行了,这里就得知道抽象类和接口从根本上解决了哪些问题.
>
> 一个类只能继承一个直接父类(可能是抽象类),却可以实现多个接口, 接口弥补了Java的单继承
>
> 抽象类为继承体系中的共性内容, 接口为继承体系中的扩展功能
>
> 接口还是后面一个知识点的基础(lambada)