

# java\_12\_13

内容：  
抽象类和接口

## Java 抽象类

### 简介

使用了关键词 **abstract** 声明的类叫作“抽象类”。如果一个类里包含了一个或多个抽象方法，类就必须指定成 **abstract**（抽象）抽象类。“抽象方法”，属于一种**不完整方法**，只含有一个声明，没有方法主体（大括号）。

### 抽象方法

使用 **abstract** 关键字修饰过的方法称之为**抽象方法**，没有方法体，只有声明的方法名称。定义的只是一组“规范”（或叫约束），就是告诉子类必须要给抽象方法提供具体的实现（子类方法实现）。

抽象方法没有定义，方法名后面直接跟一个分号，而不是花括号。

```
//抽象方法
public abstract void eat();
```

### 抽象类

**包含抽象方法的类**则称之为抽象类，通过 **abstract** 方法定义的规范，则子类**必须有**具体的实现，通过抽象类，要求约束规范子类的设计，子类之间变得通用。

### 创建抽象类

1)抽象类：

```
/**
 * 定义抽象类
 * 抽象方法
 * @author Administrator
 * @version
 */
public abstract class Animal {
    //抽象方法：没有实现，只有方法名，使用abstract关键字
    //子类继承时必须实现
    public abstract void call(); //抽象叫
```

```

    public void run(){
        System.out.println("跑了...");
    }
}

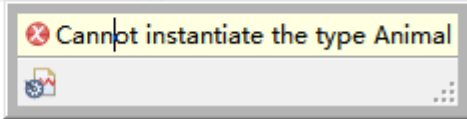
```

2).如果测试调用则不能编译（因为是抽象类）

```

1 public class AbstractTest {
2
3     public static void main(String[] args) {
4         Animal animal = new Animal();
5     }
6
7 }

```



3).通过继承抽象类及抽象方法

```

/**
 * 子类继承抽象类
 * 实现抽象方法
 * @author Administrator
 * @version
 */
public class Pig extends Animal {

    //如果没有调用call方法，则会报错，也会提示要求必须重写实现父类的抽象方法
    @Override
    public void call() {
        System.out.println("哼哼...");
    }

}

```

4)测试代码：

\*:不能实例化一个Animal类的对象，可以实例化一个Pig类对象，该对象将从 Animal类继承到成员方法(也可以通过该方法可以设置或获取成员变量)

```

/**
 * 测试抽象类
 * @author Administrator
 * @date 2019年7月7日
 * @version
 */
public class AbstractTest {

    public static void main(String[] args) {
        //抽象类不能被实例化
    }
}

```

```

//Animal animal = new Animal();
//Animal pig = new Pig();//多态（父类定义统一规范模板，子类去实现抽象方法）
Pig pig = new Pig();//继承抽象类的子类
pig.call();
}

}

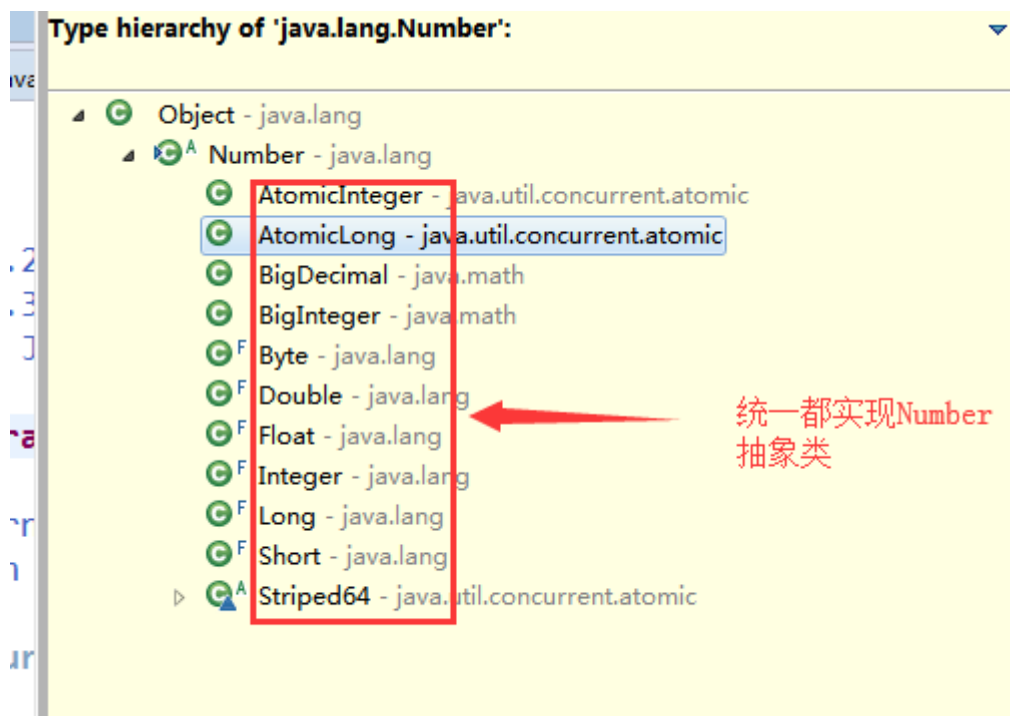
```

抽象类要点：

- 1). 有抽象方法, 类必须定义成抽象类
- 2). 抽象类不能实例化, 不能用new实例化抽象类
- 3). 抽象类可以包含属性, 普通方法, 有抽象方法, 构造方法。但构造方法也不能使用new实例, 只能被子类调用
- 4). 抽象类只能用来被子类继承

注：抽象类为子类提供一组统一组规范(模板)，子类必须实现相关的抽象方法

如：Integer类的 - 》 extends Number - 》则有很多抽象方法 intValue..., 而 Integer 则去——实现这些抽象方法。



## Java 接口

### 简介

Java 接口是一系列方法的声明，是一些方法特征的集合，一个接口只有方法的特征没有方法实现[对子类的一组约束规范]，因此这些方法可以在不同的地方被不同的类实现（子类实现具体方法），而这些实现可以具有不同的行为（功能）。

[相当于：接口方法名称定义在一个类（父类），实现接口方法体在一个类（子类），实现方法体的类可以有多个]

功能：接口实现和类继承的规则不同，为了数据的安全，继承时一个类只有一个直接父类，也就是单继承，但是一个类可以**实现多个接口**，接口弥补了类的不能多继承缺点，继承和接口的双重设计既保持了类的**数据安全**也变相**实现多继承**。

最主要的作用：一组**规范**（接口类规范[或形象理解为：父]）和具体实现方法(实现类实现方法体[或形象理解为：子])进行**分离**

\*：企业开发中多使用接口方式进行**后端层**的数据分离，Java 开发中也可以说是面向接口的设计编程

---

## 声明接口

语法格式：

```
访问修饰符权限  interface    接口名称{
public          interface    UserDao{
        抽象方法
}

接口实现类([implements 其他的接口名名] )
public  class UserDaoImpl implements UserDao {
        实现抽象类里所有的抽象方法
}
```

**Interface** 关键字用来声明一个接口。

## 创建接口

1).接口代码编写：

```
package com.fy.dao;

import com.fy.model.User;

/**
 * Dao:UserDao接口类(增删改查)
 * @author djy
 * @date 2019年1月21日
 */
public interface UserDao {

    //查询一个用户(find,get,select,que)
    public User getUser();

    //添加一个用户(save,addUser,insert)
    public int saveUser();

    //修改一个用户 (updateUser,updUser)
    public int updUser();

    //删除一个用户 (del,delete,remove,remUser)
    public int delUser();
}
```

```
}
```

特性：

- 1). 接口是隐式抽象的，当声明一个接口的时候，不必使用 `abstract` 关键字
- 2). 接口中每一个方法也是隐式抽象的，声明时同样不需要 `abstract` 关键字
- 3). 接口中的方法都是公有的，使用 `public` 定义即可
- 4). 声明定义一个接口必须使用 `interface` 关键字

描述：

- 1). 访问修饰符必须是 `public`
- 2). 接口类名，方法名：和类名及普通方法名命名规范一样
- 3). 接口关键字：`interface`
- 4). 常量：接口中属性只能是常量，只能是：`public static final`修饰，不写默认编译加上[`String name = "root";`]（一般开发中不在接口中定义常量）
- 5). 接口方法：接口方法编译器默认有[`public abstract`] `void addUser()`，如果不实现接口类，就是无用接口类
- 6). 子类只能通过 `implements` 来实现接口中的规范
- 7). 一个类实现了此接口，必须实现接口中所有的方法（遵守的一组规范），并且这些方法必须是 `public` 访问的权限
- 8). 接口类里的方法是不变化，因此极稳定，企业开发中则使用的几乎全是面向接口的设计

## 实现接口

类实现接口的时候，类要实现接口中**所有**的方法。否则，类必须声明为抽象的类。

类使用 `implements` 关键字实现接口。在类声明中，`implements` 关键字放在 `class` 声明后面。

1). 实现一个接口的代码如下：

```
package com.fy.dao;

import com.fy.model.User;

/**
 * 接口实现类
 * @author Administrator
 * @date 2019年1月21日
 */
public class UserDaoImpl implements UserDao{

    public User getUser() {
        User user = new User("张三", "江苏");
        //User user = new User();
        System.out.println("执行UserDaoImpl的方法getUser");
        //System.out.println("执行UserDaoImpl的方法getUser:输出, 查询出
        username:"+user.getUsername()+"",address:"+user.getAddress());
        return user;
    }
}
```

```

public int saveUser() {
    System.out.println("执行UserDaoImpl的方法saveUser");
    int svaeCount=1;//1为添加成功，0添加不成功
    return svaeCount;
}

public int updUser() {
    System.out.println("执行UserDaoImpl的方法updUser");
    int updCount=1;//1为修改成功，0修改不成功
    return updCount;
}

public int delUser() {
    System.out.println("执行UserDaoImpl的方法delUser:");
    int delCount=1;//1为删除成功，0删除不成功
    return delCount;
}

}

```

## 2).测试代码

```

package com.fy.test;

import com.fy.dao.UserDao;
import com.fy.dao.UserDaoImpl;
import com.fy.model.User;

/**
 * 测试User的接口实现类（四个功能）
 * @author Administrator
 * @date 2019年1月21日
 */
public class UserTest {

    public static void main(String[] args) {
        //父类接收，new 实现类
        UserDao userDao = new UserDaoImpl();//多态形势
        userDao.getUser();
        userDao.saveUser();
        userDao.updUser();
        userDao.delUser();

        System.out.println("=====子类接收，，new 实现类=====");

        //子类接收，，new 实现类
        UserDaoImpl userDaoImpl = new UserDaoImpl();
        User user = userDaoImpl.getUser();
        if(user.getUsername() !=null){

```

```

System.out.println("username:"+user.getUsername()+",address:"+user.getAddress());
    }else {
        System.out.println("查询无结果!");
    }

    int saveCount = userDaoImpl.saveUser();
    if (saveCount>0) {
        System.out.println("用户添加成功");
    }
    else {
        System.out.println("用户添加失败");
    }
    int updCount=userDaoImpl.updUser();
    System.out.println(updCount>0?"用户修改成功":"用户修改失败");
    int delNum=userDaoImpl.delUser();
    if(delNum>0){
        System.out.println("用户删除不成功");
    }else {
        System.out.println("用户删除失败");
    }
}

}

```

### 3).运行结果

```

执行UserDaoImpl的方法getUser
执行UserDaoImpl的方法saveUser
执行UserDaoImpl的方法updUser
执行UserDaoImpl的方法delUser:
=====子类接收, , new 实现类=====
执行UserDaoImpl的方法getUser
username:张三,address:江苏
执行UserDaoImpl的方法saveUser
用户添加成功
执行UserDaoImpl的方法updUser
用户修改成功
执行UserDaoImpl的方法delUser:
用户删除不成功

```

### 重写接口中声明方法的规则：

- 1). 类在实现接口的方法时，不能抛出强制性异常，只能在接口中，或者继承接口的抽象类中抛出该强制性异常
- 2). 类在重写方法时保持一致的方法名，并且应该保持相同或者相兼容的返回值类型

### 在实现接口时的一些规则：

- 1). 一个类可以同时实现多个接口。
- 2). 一个类只能继承一个类，但是能实现多个接口。
- 3). 一个接口能继承另一个接口，这和类之间的继承比较相似。

\*:开发当中使用接口，只需要写入抽象方法，implements 实现一个接口类便可，便于管理使用及排错( java 中的类只有单继承没有多继承，java 的接口类有多继承)

## 区别

- 1.普通类：具体实现方法
- 2.抽象类：一个或是一组规范（抽象方法的规范），可以有具体实现方法
- 3.接口类：一个或一组规范（不可以有实现的方法体）
- 4.抽象与接口区别：

- 1). 抽象类中的方法可以有方法体，就是能实现方法的具体功能，但是接口中的方法不行。
- 2). 抽象类中的成员变量可以是各种类型的，而接口中的成员变量只能是 public static final 类型的。（一般定义接口主要是写方法，成员变量属性都写在 model类中设置 get,set 属性）
- 3). 接口中不能含有静态代码块以及静态方法(用 static 修饰的方法)，而抽象类是可以有静态代码块和静态方法。
- 4). 一个类只能继承一个抽象类，然而一个类却可以实现多个接口

## 继承接口

一个接口可以继承另一个接口，和类之间的继承方式相似。接口的继承使用extends关键字，子接口继承父接口的方法。

Bird(小鸟)接口被Magpie和Loriot接口继承：

```
/**
 * 接口类：Bird
 * @author Administrator
 * @date 2019年1月13日
 */
public interface Bird {

    //小鸟唱歌
    public void singing(String name);

    //飞
    public void flyint(String name);
}
```

```
/**
 * 接口类继承：麻雀
 * @author Administrator
 * @date 2019年1月13日
 */
```



```
public interface Passer extends Bird {

    //喜鹊唱歌
    public void singingTwo(int num);

    //飞
    public void flyintTwo(int num);

    //吃
    public void eatTwo(String food);
}
```

```
/**
 * 接口类继承：百灵
 * @author Administrator
 * @date 2019年1月13日
 */
public interface Lark extends Bird{

    // 小鸟唱歌
    public void singingTwo();

    // 飞
    public void flyintTwo();

    // 吃
    public void eatTwo(String food);

    //喝水
    public void drinkTwo(String water);
}
```

Lark 类接口声明四个自己的方法，从Bird类接口又继承了两个方法，这样，如果某个类也实现 Lark 接口的类就有六个方法，如果别的类要是实现此接口类，就必须实现这六个方法。

相似的，实现 Passer 接口的类则实现五个方法，有两个方法来自于 Bird 接口类，因此接口类可以进行多继承方式减少代码。

## 接口多继承

Java中，类的多继承是不合法，但接口允许多继承。（因此尽量实现一次接口类便可）

接口的多继承中 extends 关键字只需要使用一次，在其后面跟着继续写上继承接口,如：

```
public interface A extends B , C...{
```

与类不同的是，接口允许多继承。

\*：尽量用一个 class 类去实现接口类进行使用,企业开发中几乎都是一个接口实现类对应一个接口类。

注：学到接口，编写 dao 层的接口类，接口实现类，业务逻辑层的接口类，接口实现类，之间层级调用（让更多同学**了解后期项目**编写的方式）

注：企业开发中接口稳定（只是接口实现类方法体变化,接口方法名称稳定，流程稳定，包名称 dao），二需求变化（逻辑流程理解,业务变化放置在业务逻辑处理层，包名称 service）