转载请注明链接: https://blog.csdn.net/feather_wch/article/details/51413541

> 介绍Java中的接口和抽象类

Java 接口和抽象类

版本号: 2018/9/29-1(22:40)

交流平台：极客窝-QQ群:779213122

> 有任何问题，欢迎交流。
> 每周末有技术交流会，只招募有分享精神和极客精神的小伙伴。
> 非诚勿扰！非诚勿扰！非诚勿扰！

点我加入极客窝!

接口(27)

1、接口是什么?

> 1. 接口-interface
> 2. 是Java语言中的一种抽象类型，是对行为的抽象，是抽象方法的集合
> 3. 接口以关键字 `interface` 来声明使用

2、接口的特点

> 1. 解耦：可以实现API定义和实现相分离
> 2. 内部方法: 要么是abstract抽象方法、要么是static静态方法，都是public修饰。(在Java 8开始，还有default method)
> 3. 内部的属性都是常量，默认使用修饰符：public static final(不用写)
> 4. 接口 只能继承 接口
> 5. 不能实例化-继承接口的 子类 可以实例化
> 6. 体现了多态，以及高内聚低耦合的特点

3、接口和类的区别

> 1. 两种不同概念
> 2. 接口使用 `interface`关键字 ； 类使用 `class`关键字
> 3. 类用于描述对象的属性和方法
> 4. 接口是类要实现的方法的抽象集合
> 5. 接口无法实例化；类可以实例化(抽象类不可以实例化)。

4、接口的使用实例

```
interface Usb{
  public void start();
  public void stop();
}
class pc implements Usb{...}

void useUsb(Usb usb){
  usb.start();
  usb.stop();
}
```

> 1. 接口中变量为 static,final int a = 0。常用作全局变量。例如：Usb.a
> 2. 接口可以继承接口
> 3. 体现了多态，以及高内聚低耦合的特点

5、接口会继承吗？(=子类是否会继承父类关于接口的实现)?

> 子类会继承父类关于接口的实现

6、接口的作用?

> 1. 接口中的变量可以作为 全局变量 来使用，例如：Usb.a
> 2. 能实现C++中多重继承的效果
> 3. 解耦

7、接口中的属性(变量)默认使用 `public static final` 修饰?

> 是的
>
> 1. 这些数据会存储在静态存储区域

8、接口中的属性(变量)可以是private的?

> 不可以!

9、接口是在多个类之间建立一种协议“protocol”?

> 是的，多个类实现了这个接口，就可以用于实现多态

Marker Interface

10、接口的职责仅仅是作为抽象方法的集合?

> 1. 不是的。
> 2. 还可以作为Marker Interface： 一种没有任何方法的接口
> 3. functional interface: Java中增加了函数式编程的支持，增加了一类定义

11、Marker interface是什么?

> 1. 标记接口
> 2. 一种没有任何方法的接口
> 3. 目的是为了声明某些东西：比如Cloneable、Serializable等

12、Marker Interface和Annotation的异同?

> 1. Marker Interface-标记接口，能够声明一些东西
> 2. Annotation-注解，也能声明一些东西。
> 3. 标记接口优点是简单直接
> 4. 注解的好处是，因为可以指定参数和值，在表达能力上要更强大。(更多人选择使用Annotation)

functional interface

13、functional interface是什么?

> 1. 函数式接口
> 2. 简单说就是一种只有一个抽象方法的接口
> 3. 通常使用 `@FunctionalInterface` 注解 来标记

> 4. Lambda 表达式本身可以看作是一类 functional interface

14、什么是函数型接口?

> 1. 一种接口，该接口内只有一个抽象方法
> 2. 该类型接口也称为 SAM接口 - Single Abstract Method Interfaces

15、functional interface的作用?

> 1. 一般用于Lambda表达式和方法引用上
> 2. 也用于熟知的 Runnable、Callable等

```
// Runnable.java
@FunctionalInterface
public interface Runnable {
    public abstract void run();
}
// Callable.java
@FunctionalInterface
public interface Callable<V> {
    V call() throws Exception;
}
```

16、@FunctionalInterface注解的作用?

> 1. Java8为函数式接口引入的新注解
> 2. 提供编译级别的错误，当编写的借口不符合函数式接口定义时，编译器会报错。

17、自定义函数式接口

> 1- 函数式接口

```
@FunctionalInterface
public interface Store {
    void sayHello(String msg);
}
```

> 2-实现函数式接口中的抽象方法

```
public class Main {
    public static void main(String[] args) throws IOException {
        // 1、实现了sayHello()方法
        Store store = (message) -> System.out.println("Hello " + message);
        // 2、调用该方法
        store.sayHello("Guest!");
    }
}
```

18、函数式接口可以包含多个抽象方法吗?

1. 不可以！违反了函数式接口的定义。

19、函数式接口中可以定义默认方法吗(default method)?

1. 可以！不违反定义。

```
@FunctionalInterface
public interface Store {
    void sayHello(String msg);

    default void doSomeMoreWork1(){
      // Method body
    }

    default void doSomeMoreWork2(){
      // Method body
    }
}
```

20、函数式接口中可以定义静态方法吗(static)?

1. 可以！作为一个具有实现的方法，不违反只能有一个抽象方法的定义。

```
@FunctionalInterface
public interface Store {
    void sayHello(String msg);

    static void staticMethod(){
      // Method body
    }
}
```

default method

21、接口中不可以有方法实现?

1. 错误。严格说，Java 8 以后，接口也是可以有方法实现的。
2. 采用 默认方法-default method

22、default method是什么

1. 默认方法
2. 允许添加到新功能到现有库的接口中，提供了一种二进制兼容的扩展已有接口的办法。
3. Java 8 开始，interface 增加了对 default method 的支持。
4. Java 9 以后，甚至可以定义 private default method。

23、为什么要有默认方法?

1. Java 8之前，接口和实现类之间的耦合度过高(tightly coupled)

> 2. 为接口添加方法，所有的实现类都必须随之修改。
> 3. 默认方法解决了该问题，可以为接口增加新方法，而不会破坏已有的接口实现。

24、default method的应用?

> 1. Java 8推出了 `Lambda` ，利用 `默认方法` ，为升级旧接口且保持向后兼容（backward compatibility）提供了途径。
> 2. Java 8中添加了一系列默认方法，主要是增加Lambda、Stream 相关的功能。

25、类似Collections之类的工具类，很多方法都适合作为 `default method` 实现在基础接口里面?

> 是的

26、实现的两个接口中有签名相同的方法该怎么办?

> 1. 需要手动解决该冲突
> 2. `InterfaceB.super.bar();InterfaceC.super.bar();` 能调用各个接口中的方法。

```
// A
interface InterfaceA {
    default void foo() {
        System.out.println("InterfaceA foo");
    }
}
// B
interface InterfaceB {
    default void bar() {
        System.out.println("InterfaceB bar");
    }
}
// C
interface InterfaceC {
    default void foo() {
        System.out.println("InterfaceC foo");
    }

    default void bar() {
        System.out.println("InterfaceC bar");
    }
}
```

> 没有冲突:

```
// 没有冲突
class ClassA implements InterfaceA, InterfaceB {
}
```

> 手动解决冲突

```
class ClassB implements InterfaceB, InterfaceC {
    @Override
    public void bar() {
        InterfaceB.super.bar(); // 调用 InterfaceB 的 bar 方法
        InterfaceC.super.bar(); // 调用 InterfaceC 的 bar 方法
        System.out.println("ClassB bar"); // 做其他的事
    }
}
```

27、抽象类可以实现这种default method的效果，为什么还需要default method?

> 1. 接口可以多重继承；抽象类不可以。
> 2. 使用defalt method，才可以在collection之类的api中增加lambda的支持。

抽象类(18)

1、什么是抽象类?

> 1. 抽象类 是 类的抽象集合
> 2. abstract关键字修饰的class
> 3. 实例

```
abstract class Animal(){};
```

2、抽象类的特点

> 1. 不能实例化
> 2. 继承方面:
>   1. 继承自抽象类的具体类，必须实现抽象类的所有抽象方法
>   2. 继承自抽象类的抽象类，可以不实现抽象类的抽象方法
> 3. 属性(成员变量方面)
>   1. 可以有变量
> 4. 方法:
>   1. 具有抽象方法的类，必须为抽象类
>   2. 抽象类可以不包含任何抽象方法。
>   3. 抽象方法一定不能是private修饰，并且不能有实现

3、抽象类的作用

> 主要为了代码重用
>
> 1. 主要是对Java类中的共有行为和属性进行抽象
> 2. 例如Collection框架中，通用部分被抽取为抽象类：AbstractList等。

4、“一个类包含abstract方法，则该类就必须为抽象类”的说法是否正确?

> 正确

5、“抽象类的子类必须要实现所有抽象方法”的说法是否正确?

> 错误!
>
> 1. 如果该子类也是抽象类，就可以不实现

6、“抽象类不可以被抽象类继承”的说法是否正确?

> 错误!

7、抽象类可以不包含任何抽象方法?

> 正确

8、抽象方法可以使用private修饰吗?

> 不可以!
>
> 1. protected
> 2. public
> 3. 默认权限

9、抽象方法可以有实现吗？既然抽象方法不可以有具体实现，为什么还有“具体子类继承抽象类时，需要实现所有抽象方法”的这种说法?

> 1. 抽象方法没有实现。是指 `abstract` 修饰的方法不能有body(具体实现)
> 2. 具体子类继承抽象类后，去实现抽象方法，本身这些方法在子类已经变成了具体方法，没有 `abstract` 关键字了。

10、包含抽象方法的类不一定要是抽象类?

> 错误！一定要是抽象类，

11、抽象类能用什么关键字修饰?不能用什么关键字修饰?

> 1. 能用关键字：abstract、public
> 2. 不能用关键字: final、private、protected、native

12、Java中类能使用哪些访问权限修饰符修饰?

> 1. 顶级类(外部类)
>   1. public
>   2. 默认
> 2. 内部类：四种权限都可以。

13、Java中类为什么不能使用protected、private修饰符修饰?

> 1. 该问题描述是有错误的。只有顶部类(外部类)才不能使用protected、private修饰符
> 2. 因为外部类的上一层是包，如果是私有的、保护的访问权限，对外不可见，失去了作为类给外部使用的意义。

14、Java中类class能否使用关键字static?

> 1. 顶级类(外部类)不可以
> 2. 内部类可以

15、抽象类可以用static修饰吗?

> 1. 内部抽象类可以
> 2. 外部顶级类不可以

16、抽象方法不能有什么关键字修饰?

> 1. final
> 2. static
> 3. private
> 4. native

17、抽象方法能用关键字 `protected` 修饰吗?

> 可以

18、子类方法继承自父类，可以在子类改变该方法的访问修饰符吗?

> 可以
>
> 1. 访问修饰符只能比父类的更大或者一致

接口与抽象类区别(10)

1、接口与抽象类区别

> 1. 方法层面:
>   1. 接口中所有的方法都不能有具体实现
>   1. 抽象类中可以有非抽象方法，因此可以有具体实现
> 2. 概念上:
>   1. 接口是对继承的一种补充。用于实现类似C++中多重继承的效果。
>   1. 接口中所有方法都不能有具体实现，是比抽象类更为抽象的内容。
>   1. 接口是“类要实现的方法的抽象集合(对行为的抽象)”
>   1. 抽象类是“类的抽象集合(对整个类的抽象，包括属性和行为)”
> 3. 接口能继承接口，不能继承类

4. 使用上:
    1. 类使用extends；接口使用implements
    1. 类能实现多个接口，但只能继承一个父类
    1. 接口能继承接口，不能继承类
5. 扩展性:
    1. 对于接口，为接口添加任何抽象方法，相应的所有实现了这个接口的类，也必须实现新增方法
    1. 对于抽象类，如果添加非抽象方法，其子类只会享受到能力扩展，而不用 担心编译出问题。

2、接口的实现类可以改变其属性吗?

> 不可以，属性是常量

3、抽象类的实现类可以改变其属性吗?

> 可以

4、类实现的不同接口中可以有同名的方法吗?

> 1. 可以有同名方法，但是要可以通过参数区分，仅仅返回值不同是不可以的
> 2. 要么名字不同

5、类实现的接口和继承的类可以有同名方法?

> 1. 可以有，需要能进行区分(参数类型不同或者参数数量不同)
> 2. 否则，只能不同名

6、抽象的定义是什么?

> 1. 将一类事物的 共同特征 总结出来并且构造出一类事物的过程
> 2. 包括: 数据抽象 和 行为抽象

7、抽象的分类

> 1. 数据抽象
> 2. 行为抽象

8、抽象的特点?

> 不会去关注 属性和行为 内部具体的细节

9、抽象的作用?

> 1. 隐藏具体的实现方式
> 2. 提供扩展的能力

10、抽象的实现方式?

> 1. 抽象类
> 2. 接口

知识扩展(25)

1、重载和重写的区别?

2、在一些情况下，需要抽象出与具体实现、实例化无关的通用逻辑，或者纯调用关系的逻辑，这时候该怎么办?

> 1. 实现由静态方法组成的工具类（Utils）
> 2. 比如 java.util.Collections。

3、java.util.Collections的作用?

> 1. 提供集合相关的工具类
> 2. 是由静态方法组成的工具类。

面向对象设计

4、面向对象的基本要素是什么?

> 1. 封装
> 2. 继承
> 3. 多态

封装

5、封装是什么？有什么用?

> 1. 封装的目的是隐藏事务内部的实现细节
> 2. 以便提高安全性和简化编程。
> 3. 封装避免了外部调用者接触到内部的细节。减少开发中无意暴露细节导致的BUG问题。比如在多线程环境暴露内部状态，导致的并发修改问题。
> 4. 从另外一个角度看，封装这种隐藏，启到简化的作用，避免太多无意义的细节浪费调用者的精力。

6、因为封装不当，导致内部细节暴露而出现BUG的场景有哪些?

> 1. 比如在多线程环境暴露内部状态，导致的并发修改内部状态的问题。

继承

7、继承是什么？有什么用?

> 1. 继承是代码复用的基础机制，
> 2. 继承是非常紧耦合的一种关系，父类代码修改，子类行为也会变动。
> 3. 在实践中，过度滥用继承，可能会起到反效果。

8、继承非常有效，所以只要能继承的地方就一定使用继承?

> 不能!
>
> 1. 继承是非常紧耦合的一种关系，父类代码修改，子类行为也会变动。
> 2. 滥用继承，可能会导致耦合度过高，而产生很多结构上的问题。

多态

9、多态是什么?多态有哪些应用场景?

> 1. 多态。在不同场景下具有不同的状态。
> 2. 多态的体现: 重写（override）和重载（overload）、向上转型
>   多态，你可能立即会想到重写（override）和重载（overload）、向上转型。简单说，重写是父子类中相同名字和参数的方法，不同的实现；重载则是相同名字的方法，但是不同的参数，本
>   质上这些方法签名是不一样的，为了更好说明，请参考下面的样例代码:

重写

10、重写是什么?

> 重写是父子类中相同名字和参数的方法，具有不同的实现

重载

11、重载是什么?

> 1. 重载则是相同名字的方法，但是不同的参数，本质上这些方法签名不一样。
> 2. 运行时根据方法签名选择合适的方法

12、如果两个方法的方法名称和参数一致，但是返回值不同，这种情况是重载吗?

> 不是！编译时会出错。
>
> 1. 但是JVM中认为是有效的重载。
> 2. 这种语义区别需要通过一个适配器

向上转型

13、向上转型是什么?

> 1. 向上转型 :子类引用的对象转换为父类类型

2. 在多态中需要将子类的引用赋给父类对象，只有这样该引用才能够具备技能调用父类的方法和子类的方法。
3. 将子类对象转为父类对象(类或者接口)

14、向上转型的注意点有哪些?

1. 向上转型时，子类单独定义的方法会丢失。无法调用子类的方法。
2. 子类引用不能指向父类对象。

15、向上转型的好处?

1. 实现多态
2. 减少重复代码，使代码变得简洁。
3. 提高系统扩展性。

S.O.L.I.D原则

16、 设计原则S.O.L.I.D是指什么?

1. 单一职责（Single Responsibility）
2. 开关原则（Open-Close, Open for extension, close for modification）
3. 里氏替换（Liskov Substitution）
4. 接口分离（Interface Segregation）
5. 依赖反转（Dependency Inversion）

17、S-单一职责原则是什么?

1. Single Responsibility
2. 类或者对象最好是只有单一职责，在程序设计中如果发现某个类承担着多种义务，可以考虑进行拆分。

18、O-开关原则是什么?

1. Open-Close, Open for extension, close for modification
2. 设计要对扩展开放，对修改关闭。
3. 程序设计中药尽量避免因为新增功能而修改已有实现。

19、L-里氏替换是什么?

1. Liskov Substitution
2. 面向对象的基本要素之一
3. 进行继承关系抽象时，凡是可以用父类的地方，都可以用子类替换。

20、I-接口分离是什么?

1. Interface Segregation

2. 类和接口设计时，将具有过多方法的接口，拆分为功能单一的多个接口，将行为进行解耦。
3. 在未来维护中，如果某个接口设计有变，不会对使用其他接口的子类构成影响。

21、接口分离原则是解决什么问题的?

1. 我们在进行类和接口设计时，如果在一个接口里定义了太多方法，其子类会出现只有部分方法对它是有意义的，这就破坏了程序的 内聚性 。
2. 对于这种情况，可以通过拆分为多个功能单一的接口，将行为进行解耦。

22、D-依赖反转是什么?

1. Dependency Inversion
2. 实体应该依赖于抽象而不是实现。
3. 高层次模块，不应该依赖于低层次模块，而是应该基于抽象。
4. 该原则优势是进行了高度解耦。

实践中的OOP原则

23、实践中，是否要严格遵循SOLID原则?

1. 不是
2. 例如 `Java 10` 引入了 本地方法类型推测 和 `var`类型
3. 根据 里氏替换原则 ，需要这样定义变量: `List<String> list = new ArrayList<>();`
   而 `var list = new ArrayList<String>();` 中，list会被推测为 `ArrayList<String>` 类型。
4. 这种语法上的便利，增强了程序对实现的依赖。微小的类型泄露，提高了书写的便利性和代码可读性。
5. 需要按照具体场景具体分析。

24、为什么Java10引入的本地方法类型推测和var类型，违背了里氏替换原则?

1.

25、面试题，下面代码可以用哪些OOP设计原理进行改进?

```
public class VIPCenter {
  void serviceVIP(T extend User user>) {
    if (user instanceof SlumDogVIP) {
        // 穷 X VIP，活动抢的那种
        // do somthing
    } else if(user instanceof RealVIP) {
        // do somthing
    }
    // ...
  }
}
```

> 1. 违反了开关原则。如果有新的会员、新的服务方式，会需要去修改已有代码，导致有可能错误影响到其他的代码。
> 2. 这是一种将简单工厂模式，变成了工厂方法模式?

```
// VIP中心，根据服务商提供的服务，来服务用户。
public class VIPCenter {
  private Map<User.TYPE, ServiceProvider> providers;
  void serviceVIP(T extend User user） {
      providers.get(user.getType()).service(user);
  }
}
// 服务商的接口
interface ServiceProvider{
  void service(T extend User user) ;
}
// 具体服务-1
class SlumDogVIPServiceProvider implements ServiceProvider{
  void service(T extend User user){
    // do somthing
  }
}
// 具体服务-2
class RealVIPServiceProvider implements ServiceProvider{
  void service(T extend User user) {
    // do something
  }
}
```

问题汇总

参考资料

1. 第13讲 | 谈谈接口和抽象类有什么区别?
2. functional-interfaces
3. 什么是函数式接口
4. 了解 lambda
5. Java 8 默认方法

交流平台：极客窝-QQ群:779213122

> 有任何问题，欢迎交流。
> 每周末有技术交流会，只招募有分享精神和极客精神的小伙伴。
> 非诚勿扰！非诚勿扰！非诚勿扰！

点我加入极客窝!

极客窝