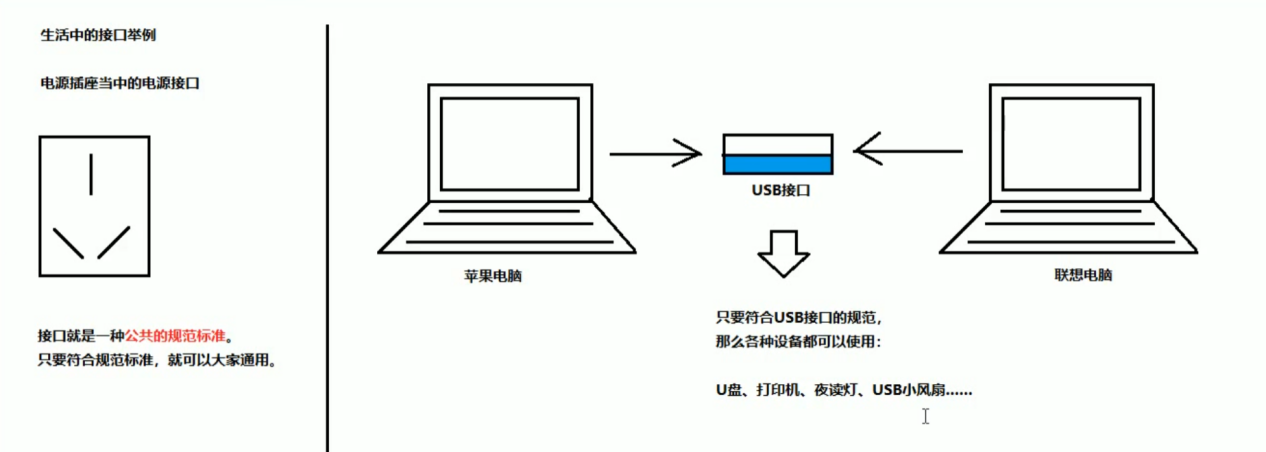
* + - * 1. 接口概述与生活举例



* + - * 1. 接口的定义基本格式

*/\*  
接口就是多个类的公共规范。  
接口是一种引用数据类型，最重要的内容就是其中的：抽象方法。  
  
如何定义一个接口的格式：  
public interface 接口名称{  
 // 接口内容  
}  
备注：换成了关键字interface之后，编译生成的字节码文件仍然是： java --> .class。  
  
如果是Java 7 ，那么接口中可以包含的内容有：  
1. 常量  
2. 抽象方法  
  
如果是Java 8 ，还可以额外包含：  
3. 默认方法  
4。 抽象方法  
  
如果是Java 9 ， 和可以额外包含：  
5. 私有方法  
 \*/*

* + - * 1. 接口的抽象方法定义

*/\*  
在任何版本的Java中，接口都能定义抽象方法：  
格式：  
public abstract 返回值类型 方法名称（参数列表）;  
  
注意事项：  
1. 接口当中的抽象方法，修饰符必须是两个固定的关键字，public abstract。  
2. 这两个关键字修饰符，可以选择性的省略。  
3. 方法的三要素：可以随机定义。  
  
接口的使用步骤：  
1. 接口不能直接使用，必须有一个“实现类”来“实现”该接口。  
格式：  
public class 实现类名称 implement 接口名称{  
 // ...  
}  
2. 接口的实现类必须覆盖重写（实现）接口中所有的抽象方法。  
实现：去掉abstr关键字，加上方法体大括号。  
3. 创建实现类的对象，进行使用。  
  
注意事项：  
如果实现类并没有覆盖重写接口中所有的抽象方法，那么这个实现类自己就必须是抽象类。  
 \*/*

接口(其中 public abstract 可以省略）：

**public interface** MyInterfaceAbstract {  
  
 *// 这是一个抽象方法* **public abstract void** methodAbs1();  
  
 *// 这也是抽象方法* **abstract void** methodAbs2();  
  
 *// 这也是抽象方法* **public void** methodAbs3();  
  
 *// 这也是一个抽象方法* **void** methodAbs4();  
}

子类：

**package** demo04;  
  
**public class** MyInterAbstractlmp **implements** MyInterfaceAbstract {  
 @Override  
 **public void** methodAbs1() {  
 System.***out***.println(**"这是第一个方法！"**);  
 }  
 @Override  
 **public void** methodAbs2() {  
 System.***out***.println(**"这是第二个方法！"**);  
 }  
 @Override  
 **public void** methodAbs3() {  
 System.***out***.println(**"这是第三个方法！"**);  
 }  
 @Override  
 **public void** methodAbs4() {  
 System.***out***.println(**"这是第四个方法！"**);  
 }  
}

主函数：

public class Demo01Interface {  
 public static void main(String[] args) {  
  
  
 // 错误写法！不能直接new接口对象使用。  
 // MyInterfaceAbstract inter = new MyInterfaceAbstract();  
  
 // 创建一个实现类的对象使用  
 MyInterfaceAbstractlmpl impl = new MyInterfaceAbstractlmpl();  
 impl.methodAbs1();  
 impl.methodAbs2();  
 impl.methodAbs3();  
 }  
}

* + - * 1. 接口的默认方法定义

/\*  
从Java 8 开始，接口哩允许定义默认方法。  
格式：  
public default 返回值类型 方法名称（参数列表）{  
 方法体  
}  
  
备注：接口当中的默认方法，可以解决接口升级的问题。  
  
接口的默认方法，可以通过接口实现类对象，直接调用。下边的AAA  
接口的默认方法，也可以被接口实现类进行覆盖重写。 下边的BBB

注意：空代码也可以覆盖以前的代码。

@Override  
public void methodDefault() {  
}

最后主函数输出的为kong，而不是接口中的内容。  
 \*/

默认接口：

public interface MyInterdaceDefault {  
  
 // 抽象方法  
 public abstract void methodAbs();  
  
 // 新添加一个抽象方法  
// public abstract void methodAbs2();  
  
 // 新添加的方法，改成默认方法  
 public default void methodDefault(){  
 System.out.println("这是一个新添加的默认方法");  
 }  
}

继承方法A：

package demo;  
public class MyInterfaceDefaultA implements MyInterdaceDefault {  
 @Override  
 public void methodAbs() {  
 System.out.println("实现了抽象方法，AAA");  
 }  
}

继承方法B：

package demo;  
public class MyInterfaceDefaultB implements MyInterdaceDefault {  
 @Override  
 public void methodAbs() {  
 System.out.println("实现了抽象方法，BBB");  
 }  
 @Override  
 public void methodDefault() {  
 System.out.println("实现类B覆盖重写了接口的默认方法");  
 }  
}

主函数：

package demo;  
public class Demo01Interface {  
 public static void main(String[] args) {  
 // 创建了实现类对象  
 MyInterfaceDefaultA a= new MyInterfaceDefaultA();  
 // 调用抽象方法，实际运行的是右侧的实现类。  
 a.methodAbs(); //实现了抽象方法，AAA  
  
 // 调用默认方法，如果实现类当中没有，会向上找接口  
 a.methodDefault(); // 这是一个新添加的默认方法  
  
 MyInterfaceDefaultB b = new MyInterfaceDefaultB();  
 b.methodAbs(); // 实现了抽象方法，BBB  
 b.methodDefault(); // 实现类B覆盖重写了接口的默认方法  
  
 }  
}

* + - * 1. 接口静态方法的定义

/\*  
从Java 8 开始，接口当中允许定义静态方法。  
格式：  
public static 返回值类型 方法名称（参数列表）{  
 方法体  
}  
  
提示：就是将abstract或者default换成static即可，带上方法体。  
 \*/

接口：

public interface MyInterfaceStatic {  
  
 public static void methodStatic(){  
 System.out.println("这是接口的静态方法");  
 }  
}

定义一个继承：

package demo01;  
  
public class MyInterfaceStaticImpl implements MyInterfaceStatic{

}

注意：这个静态方法用不到这个继承

主函数：

package demo01;  
  
public class Demo03Interface {  
 public static void main(String[] args) {  
 // 创建了实现类对象  
 MyInterfaceStaticImpl impl = new MyInterfaceStaticImpl();  
  
 // 错误写法  
// impl.methodStatic();  
  
 // 直接通过接口名称调用静态方法  
 MyInterfaceStatic.methodStatic(); // 这是接口的静态方法  
 }  
}

* + - * 1. 接口的私有方法定义

package demo03;  
  
/\*  
问题描述：  
我们需要抽取一个共有方法，用来解决两个默认方法之间重复代码的问题。  
但是这个共有方法不应该让实现类使用，应该是私有化的。  
  
解决方案：  
从Java 9开始，接口当中允许定义私有方法。  
1. 普通私有方法，解决多个默认方法之间重复代码问题  
格式：  
private 返回值类型 方法名称（参数列表）{  
 方法体  
}  
  
2. 静态私有方法，解决多个静态方法之间重复代码问题  
格式：  
private static 返回值类型 方法名称（参数列表）{  
 方法体  
}  
 \*/

接口：  
public interface MyInterfacePrivateA {  
  
 public default void methodDefault1(){  
 System.out.println("默认方法1");  
 methodCommon();  
 }  
  
 public default void methodDefault2(){  
 System.out.println("默认方法2");  
 methodCommon();  
 }  
  
 public default void methodCommon(){  
 System.out.println("AAA");  
 System.out.println("BBB");  
 System.out.println("CCC");  
 }  
}

接口常量的定义和使用接口当中也可以定义“成员变量”，但是必须使用public static final三个关键字修饰。  
从效果上看，这其实就是接口的【常量】。  
格式：  
public static final 数据类型 常量名称 = 数据值;  
备注：  
一旦使用final关键字进行修饰，说明不可以改变。  
  
注意事项：  
1. 接口当中的常量，可以省略public static final，注意：不写也照样是这样。  
2. 接口当中的常量，必须进行赋值，不能不赋值。  
3. 接口当中常量的名称，使用完全大写的字母，用下划线进行分隔。（推荐命名规则）  
 \*/

类：

public interface MyInterfacePrivateA {  
  
 // 其实就是一个常量，一旦赋值，不可以在就行修改  
 public static final int NUM\_OF\_MY\_CLASS = 12;  
}

主函数：

public class zhu {  
  
 public static void main(String[] args) {  
 // 访问接口当中的常量  
 System.out.println(MyInterfacePrivateA.NUM\_OF\_MY\_CLASS);  
 }  
}

* + - * 1. 接口内容小结

在Java 9+版本当中，接口的内容可以有：

1. 成员变量其实就是常量，格式：

[public] [static] [final] 数据类型 常量名称 = 数据值；

注意： 1. 常量必须进行赋值，而且一旦赋值不能在改变。

2. 常量名称完全大写，用下划线进行分隔。

2. 接口中最重要的就是抽象方法，格式：

[public] [abstract] 返回值类型 方法名称（参数列表）;

注意：实现了必须覆重写接口所有的抽象方法，除非实现类就是抽象类。

1. 从Java 8开始，接口里允许定义默认方法，格式：

[public] default 返回值类型 方法名称（参数列表）{方法体}

注意：默认方法也可以被覆盖重写

1. 从Java 8 开始，接口里允许定义静态方法，格式：

[public] static 返回值类型 方法名称（参数列表）{方法体}

注意：应该通过接口名称进行调用，不能通过实现类对象调用接口静态方法

1. 从Java 9开始，接口里允许定义私有方法，格式：

普通私有方法：private 返回值类型 方法名称（参数列表）{方法体}

静态私有方法：private static 返回值类型 方法名称（参数列表）{方法体}

注意：private的方法只有接口自己才能调用，不能被实现类或别人使用。

* + - * 1. 继承父类并实现多个接口
        2. /\*  
           使用接口的时候，需要注意：  
             
           1. 接口是没有静态代码块或者构造方法。  
           2. 一个类的直接父类是唯一的，但是一个类可以同时实现多个接口。  
           格式：  
           public class MyInterfaceImpl implements MyInterfaceA,MyInterfaceB {  
            // 覆盖重写所有抽象方法  
           }  
           3. 如果实现类所实现的多个接口当中，存在重复的抽象方法，那么只需要覆盖重写一次即可。  
           4. 如果实现类没有覆盖重写所有接口当中的抽象方法，那么实现类必须就是一个抽象类。  
           5. 如果实现类所实现的多个接口当中，存在重复的默认方法，那么实现类一定要对冲突的默认方法进行覆盖重写。  
           6. 一个类如果直接父类当中的方法，和接口当中的默认方法产生了冲突，优先用父类当中的方法。  
            \*/

默认接口：

public interface MyInterface {  
  
 public default void method(){  
 System.out.println("接口默认的方法");  
 }  
}

接口A：

public interface MyInterfaceA {  
  
 public abstract void methodA();  
  
 public abstract void methodAbs();  
  
 public default void methodDefault(){  
 System.out.println("默认方法AAA");  
 }  
}

接口B：

public interface MyInterfaceB {  
  
// static {  
  
// } 错误写法，接口不能有静态代码块  
  
// public MyInterfaceA(){  
// } 错误写法，接口不能有构造方法。  
 public abstract void methodB();  
  
 public abstract void methodAbs();  
  
 public default void methodDefault(){  
 System.out.println("默认方法BBB");  
 }

以上三个接口的抽象类：

public abstract class MyInterfaceAbstract implements MyInterfaceA,MyInterfaceB{  
 @Override  
 public void methodA() {  
 }  
 @Override  
 public void methodB() {  
 }  
 @Override  
 public void methodAbs() {  
 }  
 @Override  
 public void methodDefault() {  
 }  
}

以上三个接口的覆盖重写：  
public class MyInterfaceImpl /\* extends Object\*/ implements MyInterfaceA,MyInterfaceB {  
 @Override  
 public void methodA() {  
 System.out.println("覆盖重写A方法");  
 }  
 @Override  
 public void methodB() {  
 System.out.println("覆盖重写B方法");  
 }  
 @Override  
 public void methodAbs() {  
 System.out.println("覆盖重写AB接口都用的抽象方法");  
 }  
 @Override  
 public void methodDefault() {  
 System.out.println("对多个接口当中发生冲突的默认方法进行了覆盖重写");  
 }  
}

子类：

public class Zi extends Fu implements MyInterface {  
}

父类：

public class Fu {  
  
 public void method(){  
 System.out.println("父类方法");  
 }  
}

主函数：

public class zhu {  
 public static void main(String[] args) {  
  
 Zi zi = new Zi();  
 zi.method();  
 }  
}

* + - * 1. 接口之间的多继承

1. 类与类之间是单继承关系，直接父类至于一个。
2. 类与类之间是多实现的，一个类可以实现多个接口。
3. 接口与接口之间是多继承的。

注意事项：

1. 多个父接口当中的抽象方法如果重复，没关系。
2. 多个父接口当中的默认方法如果重复，那么子接口必须进行默认方法的覆盖重写。

【而且带着default关键字】