

## 继承，接口，多态：

### 继承相关：

继承语法：

class 类名称 extends 类名称；

继承后可直接调用父类中的方法，实现代码的高效运行。

```
package com.Dish.java;

public class Dish_0 { 1个用法 2个继承者 新 *
    public String name;
    public double price; 1个用法
    public void profile() { 0个用法 2个重写 新 *
        System.out.println("番茄炒蛋是绝好的菜肴，，只需简单翻炒便可还原原汁美味");
    }
    Dish_0(String name, double price) { 1个用法 新 *
        this.name = "番茄炒蛋";
        this.price = 10.0;
    }
    Dish_0() { 0个用法 新 *
    }
}
```

```
package com.Dish.java;

public class Dish_1 extends Dish_0{ 0个用法 新 *
    Dish_1(String dishName,double price){ 0个用法 新 *
        super(dishName,price);
        this.name = "佛跳墙";
        this.price = 68;
    }
    public void profile() { 0个用法 新 *
        System.out.println("佛跳墙采用多种名贵材料，经过漫长的工序，打造成了一");
    }
}
```

```

package com.Dish.java;

public class Dish_2 extends Dish_0{ 0个用法 新*
    public Dish_2(String name , double price) { 0个用法 新*
        super();
        this.name = "扬州炒饭";
        this.price = 20;
    }
    public void profile() { 0个用法 新*
        System.out.println("扬州炒饭味道鲜美，集多种素材材料于一体，菜品和米饭
    }
}

```

## 二. 接口和多态相关:

接口是什么? 接口就是多个类的公共规范: 接口是一种引用数据类型。

定义接口格式: public interface 接口名称;

但接口不能直接用来创建对象, 需要一个实现类来实现;

格式为 class 实现类名称 implements 接口名称;

接口中可以创立抽象方法, 默认方法, 静态方法, 私有方法, 在实现类中抽象方法要覆盖重写。

### 多态是什么:

一个对象拥有多种形态, 这实际上就是对象的多态性;

语法格式: 父类 对象名 = new 子类;

把子类对象当作父类来使用。

以下为我在 Order 接口中设置的两个 cook 方法和 check 方法, 并分别让两个菜品类继承了。

```

public interface Order { 11个用法 3个实现 新* 3个相关问题
    public default void cook() { 0个用法 新*
        System.out.println("按照profile中介绍进行烹饪");
    }
    boolean check(boolean hasIngredients); 0个用法 3个实现 新*
    public default void getDishes() { 1个用法 1个重写 新*
    }

public class Dish_1 extends Dish_0 implements Order{ 1个用法 新*
    Dish_1(String dishName,double price){ 1个用法 新*
        super(dishName,price);
        this.name = "佛跳墙";
        this.price = 68;
    }
    public void profile() { 0个用法 新*
        System.out.println("佛跳墙采用多种名贵材料，经过漫长的工序，打造成了一道

    @Override 0个用法 新*
    public boolean check(boolean hasIngredients) {
        return hasIngredients;
    }
}

```

此后会优先访问实现类中重写的方法，对于继承了接口的类而言，引用类中的方法时会向接口去寻找。


以下我为 System 函数进行的改写

```

public class System { 新*
    static PrintStream out; 8个用法
    private Map<String,Integer> inventory = new HashMap<>(); 2个用法
    public void ManageOrder(List<Order> orders) { 0个用法 新*
        inventory = new HashMap<>();
        Order order = new Order() { 新*

```

```
        if(checkInventory(orders)) {
            int orderID = generateOrderID();
            PrintOrder((com.Dish.java.Order) order);
        } else {
            System.out.println("原料不足, 订单");
        }
    }
}
```

 com.Dish.java  
public interface Order  
 java1

```
private int generateOrderID() { 1个用法 新 *
    return 1;
}
private int getOrderID() { 0个用法 新 *
    return 1;
}

@ private boolean checkInventory(List<Order> orders) { 1个用法 新 *
    for(Order order : orders) {
        String dishName = order.getDishName();
        Integer stock = inventory.get(dishName);
        if (stock!= null && stock >= order.getQuantity()) {
            return true;
        }
    }

    return false;
}

@ private void PrintOrder(Order order) { 1个用法 新 *
    System.out.println( "订单编号" +order.getOrderID());
    System.out.println( "菜品" +order.getDishName());
}

}
```

相应的解释：

首先定义了一个 ManageOrder 的方法，里面接受以 Order 列表的参数

然后创建了一个空的哈希映射用于存储键值对数据

引用了一个 checkInventory (orders) 的方法在图二中有体现

New 了 order 对象出来，利用图二体现的 PrintOrder 打出对应的信息，否则输出“原料

不足，订单取消”。

## 泛型

### 什么是泛型？

泛型提供了编译类型安全监测机制，允许在编译时检测到非法数据类型结构。泛型本质就是参数化类型。

泛型有诸多好处，[以下为 ManageOrder 的补全：](#)

```
class TableCustomer { 3个用法 新 *
    public int tableID; 1个用法
    public int customerID; 0个用法
    public void getTableID() { 1个用法 新 *
        System.out.println("Table ID: " + tableID + "把餐送去相应的桌子");
    }
}

class WechatCustomer { 4个用法 新 *
    public String address; 1个用法
    public boolean takeout; 2个用法
    public boolean takeout() { 0个用法 新 *
        return takeout;
    }
    public void deliver() { 1个用法 新 *
        if(takeout) {
            System.out.println("食品已经打包，正在送往地址: " + address);
        }else {
            System.out.println("顾客未选择打包，订单已经完成");
        }
    }
}
```

```
public void ManageOrder(List<Order> orders ,List<TableCustomer> custo
    TableCustomer custom1 = new TableCustomer() ;
    WechatCustomer custom2 = new WechatCustomer() ;
    custom1.getTableID();
    custom2.deliver();
}
```

顾客订单处理系统更新完毕!