Java 实验报告

徐云凯 1713667

已编译的 JavaDoc

问题分析

一． 整体结构

项目需要能够针对不同的套餐使用不同的具体处理方式，但是需要共用同一套方法调用，故应当为每一个套餐设置一个类并在其中实现所有套餐间有差异的方法。所有套餐类共同继承自同一父类 ServicePackage，从而利用多态实现灵活调用。而不同的套餐具有重复内容，可能使用到同一方法，故使用接口实现不同的类调用类似方法，提高代码重用性。此外，项目要求了大量的菜单页与具体功能性应用，不适合分散到各个类，故使用一个单独的工具类 CardUtil，整合大多数工具方法。

另外，对于项目要求的各个一级二级菜单，在主类中建立单独的静态方法放置一级二级菜单代码，菜单只进行对工具类的调用而不进行具体实现，从而实现了 main 函数-菜单方法-工具类-底层类具体功能实现 的四级层次分离。最大程度上提高了代码的可复用与修改的灵活性。

故依据以上要求，项目各个类与接口结构如下：

程序包com.sosoMobie的分层结构

类分层结构

- java.lang.Object
  - com.sosoMobie.**CardUtil**
  - com.sosoMobie.**ConsumInfo**
  - com.sosoMobie.**Main**
  - com.sosoMobie.**MobileCard**
  - com.sosoMobie.**Scene**
  - com.sosoMobie.**ServicePackage**
    - com.sosoMobie.**NetPackage** (implements com.sosoMobie.NetService)
    - com.sosoMobie.**SuperPackage** (implements com.sosoMobie.CallService, com.sosoMobie.NetService, com.sosoMobie.SendService)
    - com.sosoMobie.**TalkPackage** (implements com.sosoMobie.CallService, com.sosoMobie.SendService)
  - java.lang.Throwable (implements java.io.Serializable)
    - java.lang.Exception
      - com.sosoMobie.**InsufficientBalanceException**

接口分层结构

- com.sosoMobie.**CallService**
- com.sosoMobie.**NetService**
- com.sosoMobie.**SendService**

其中,除了在项目中要求的几个类之外,自定义了 InsufficientBalanceException 异常类,用于在处理消费场景时可能遇到的余额不足异常。

二. 各功能实现

(按项目中菜单项次序排列)

1. 一级二级菜单

1.1. 使用 do...while 循环反复执行对菜单项目的选择

内部使用 switch-case 语句选择选项执行,对于非法输入使用 try-catch(InputMismatchException e)处理,并新建 scanner 对象以刷新输入缓冲区

```
private static void mainMenu(CardUtil cardUtil){
    Scanner scanner = new Scanner(System.in);
    int select = -1;

    do{
        System.out.print("\n**********欢迎使用嗖嗖移动业务大厅**********\n" +
                "1.用户登录 2.用户注册 3.使用嗖嗖 4.话费充值 5.资费说明 6.退出系统\n" +
                "请选择:");
        try {
            select = scanner.nextInt();
        }
        catch(InputMismatchException e){
            scanner = new Scanner(System.in);
            System.out.println("请输入正确的数字");
        }

        switch (select)
        {...}
    } while(true);  //循环执行主要方法体
}
```

1.2. 一级菜单 第一选项

逐层验证卡号密码，一旦不正确就返回重新输入，都验证成功后调用二级菜单方法。

```
case 1:
    String card, password;
    boolean flag;

    System.out.print("请输入手机卡号：");
    card = scanner.next();
    flag = cardUtil.isExistCard(card);
    if(flag){   //先检查卡号是否存在，验证存在后再输入密码
        System.out.print("请输入密码：");
        password = scanner.next();
        flag = cardUtil.isExistCard(card, password);
        if(flag){   //再检查密码是否正确
            System.out.println("登录成功");
            subMenu1(cardUtil, card);   //进入二级菜单
        }
        else{
            System.out.println("密码输入错误，请重试");
        }
    }
    else{
        System.out.println("手机卡号不存在，请重试");
    }
    break;
```

1.3. 一级菜单 第二选项

调用 CardUtil 类对应方法生成新卡号，并格式化输出，输入时检查输入合法性。

```
case 2:
    MobileCard newCard = new MobileCard();
    int numIndex = -1, numberCount = 9;

    System.out.println("*****可选的卡号*****");
    String[] numbers = cardUtil.getNewNumbers(numberCount);
    int cntLine = 0;    //该变量用于计数卡号数量，决定是否换行
    for(int i = 0; i < numbers.length; i++, cntLine++){
        if(cntLine != 0)
            System.out.print("\t");
        System.out.print(i + 1 + "." + numbers[i]);
        if(cntLine == 2){
            cntLine = -1;
            System.out.println();
        }
    }
    do{
        System.out.print("请选择卡号(输入1~9的序号)：");
        try{
            numIndex = scanner.nextInt();
        }
        catch (InputMismatchException e){
            numIndex = -1;
        }
    } while(numIndex < 1 || numIndex > numberCount);    //循环检查输入是否合法
    newCard.cardNumber = numbers[numIndex];
```

```
do{
    System.out.print("1.话痨套餐 2.网虫套餐 3.超人套餐， 请选择套餐(输入序号)：");
    try{
        numIndex = scanner.nextInt();
    }
    catch (InputMismatchException e){
        numIndex = -1;
    }
} while(numIndex < 1 || numIndex > 3);    //循环检查输入是否合法
if(numIndex == 1)   //通话、上网、超级套餐分别代号1、2、3
    newCard.serPackage = new TalkPackage();
else if(numIndex == 2)
    newCard.serPackage = new NetPackage();
else
    newCard.serPackage = new SuperPackage();

System.out.print("请输入姓名：");
newCard.userName = scanner.next();

System.out.print("请输入密码：");
newCard.password = scanner.next();

double preMoney = -1;
boolean tempFlag = true;
do{
    System.out.print("请输入预存话费金额：");
    if(!tempFlag)
        System.out.print("您预存的话费金额不足以支付本月固定套餐资费，请重新充值：");
    try{
        preMoney = scanner.nextDouble();
    }
    catch (InputMismatchException e){
        System.out.println("请输入正确的数字");
    }
    tempFlag = ! (preMoney < newCard.serPackage.price) || ! (preMoney > 0);
} while(preMoney <= 0);    //循环检查输入是否合法
```

随后输入姓名密码预存话费等信息，输入时除 string 类型外均检查合法性。

最后直接扣除首月月费，存入后台。

```
//直接扣除首月套餐费再存入
newCard.money = preMoney - newCard.serPackage.price;

cardUtil.addCard(newCard);
System.out.print("注册成功！");
newCard.showMeg();
newCard.serPackage.showInfo();

break;
```

1.4. 一级菜单 第三选项

直接调用 useSoso 方法

```
case 3:
    String num;
    do{
        System.out.print("请输入手机卡号：");
        num = scanner.next();
        if(!cardUtil.isExistCard(num))
            System.out.println("手机卡号不存在，请重试");
        else
            break;
    } while(true);
    cardUtil.useSoso(num);
    break;
```

1.5. 一级菜单 第四选项

输入并检查合法性后，调用工具类的 chargeMoney 方法

```
case 4:
    System
    String
    if(car
        double mount = -1;
        do{
            System.out.print("请输入充值金额：");
            try{
                mount = scanner.nextDouble();
            }
            catch (InputMismatchException e){
                System.out.println("请输入正确的数字");
            }
        } while(mount <= 0);    //循环检查输入是否合法

        if (mount < 50){
            System.out.println("充值金额至少为50元");
        }
        else{
            cardUtil.chargeMoney(number, mount);
        }
    }
    else{
        System.out.println("充值卡号不存在，请重试");
    }
    break;
case 5:
    cardUtil.showDescription();
    break;
```

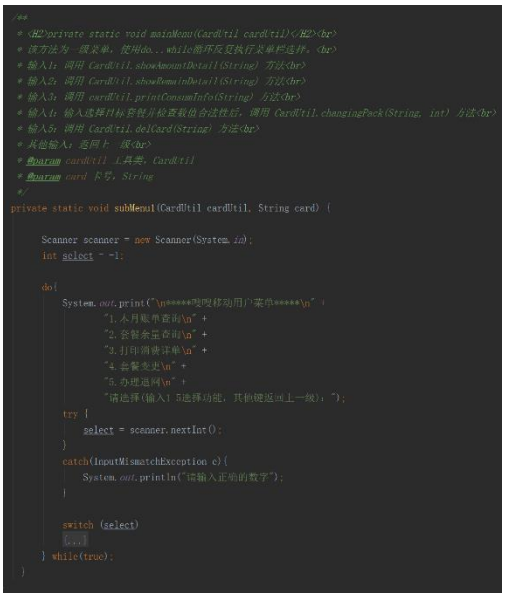
1.6. 一级菜单 第五选项

调用工具类方法

1.7. 一级菜单 第六选项及其他输入

```
case 6:
    System.out.println("感谢使用嗖嗖移动业务大厅"); //退出
    return;

default:
    System.out.println("请在1-6中做出选择");    //对于任意其他输入，显示提示信息后重新输入
```

1.8. 二级菜单



```
/**
 * <H2>private static void mainMenu(CardUtil cardUtil)</H2><br>
 * 该方法为一级菜单，使用do...while循环反复执行菜单栏选择。<br>
 * 输入1：调用 CardUtil.showAmountDetail(String) 方法<br>
 * 输入2：调用 CardUtil.showRemainDetail(String) 方法<br>
 * 输入3：调用 cardUtil.printConsumInfo(String) 方法<br>
 * 输入4：输入选择目标套餐并检查数值合法性后，调用 CardUtil.changingPack(String, int) 方法<br>
 * 输入5：调用 CardUtil.delCard(String) 方法<br>
 * 其他输入：返回上一级<br>
 * @param cardUtil 工具类，CardUtil
 * @param card 卡号，String
 */
private static void subMenu(CardUtil cardUtil, String card) {

    Scanner scanner = new Scanner(System.in);
    int select = -1;

    do{
        System.out.print("\n*****嗖嗖移动用户菜单*****\n" +
                "1.本月账单查询\n" +
                "2.套餐余量查询\n" +
                "3.打印消费详单\n" +
                "4.套餐变更\n" +
                "5.办理退网\n" +
                "请选择(输入1-5选择功能，其他键返回上一级)：");
        try {
            select = scanner.nextInt();
        }
        catch(InputMismatchException e){
            System.out.println("请输入正确的数字");
        }

        switch (select)
        {...}
    } while(true);
}
```

类似一级菜单循环方式

1.9. 二级菜单功能

分别调用工具类对应方法

```
switch (select)
{
    case 1:
        cardUtil.showAmountDetail(card);
        break;

    case 2:
        cardUtil.showRemainDetail(card);
        break;

    case 3:
        cardUtil.printConsumInfo(card);
        break;

    case 4:
        System.out.println("******套餐变更******");
        String numIndex;
        do{
            System.out.print("1.话痨套餐 2.网虫套餐 3.超人套餐 请选择(序号)：");
            numIndex = scanner.next();
        } while(!numIndex.equals("1") && !numIndex.equals("2") && !numIndex.equals("3"));

        cardUtil.changingPack(card, numIndex);

        break;

    case 5:
        cardUtil.delCard(card);
        return;

    default:
        return;
}
```

2. 工具类

2.1. 构造函数与初始化

构造函数初始化所有列表并载入预置的账号。构造函数的最后调用 initScenes() 函数初始化 List<Scene> scenes 完成对场景的初始化

2.2. 判断卡号是否存在及卡号密码是否正确

两个函数功能接近，使用函数重载。前者直接使用 Array 的 containsKey 方法查询卡号，后者使用 get 检查正确密码是否符合传入的密码参数。

```
/***
 *  isExistCard
 *［判断指定卡号是否存在于系统中］
 * @param number: 卡号
 * @return boolean:存在返回true，不存在返回false
 * @author  徐云凯
 */

public boolean isExistCard(String number) { return cards.containsKey(number); }

/***
 *  isExistCard
 *［判断指定卡号是否存在且密码是否正确］
 * @param number: 卡号
 * @param password: 密码
 * @return boolean:卡号存在且密码正确返回true，其他返回false
 * @author  徐云凯
 */

public boolean isExistCard(String number, String password) {
    if (isExistCard(number)) return cards.get(number).password.equals(password);
    else return false;
}
```

2.3. 获取新卡号

使用 random.nextInt(int)获取不重复的新号码，检查不重复之后加入输出数组中。

```
public String[] getNewNumbers(int count) {
    Random random = new Random();
    int num;
    String[] numbers = new String[count];
    for (int i = 0; i < count; i++){
        do {
            num = random.nextInt( bound: 100000000);
        } while (isExistCard(String.valueOf(13900000000L + num)));
        numbers[i] = String.valueOf(13900000000L + num);
    }
    return numbers;
}
```

2.4. 添加与删除

调用 array 的 put,remove 方法实现增加删除

```
public void addCard(MobileCard card) { cards.put(card.cardNumber, card); }

/***
 *  delCard
 *［从系统中删除电话卡］
 * @param card: 需要删除的电话卡卡号
 * @author  徐云凯
 */

public void delCard(String card) {
    cards.remove(card);
    System.out.print("*****办理退网*****\n" + "卡号" + card + "办理退网成功！\n" + "谢谢使用！\n");
}

/***
 *  showRemainDetail
 *［显示电话卡的套餐余量］
 * @param number: 需要查询的卡号
 * @author  徐云凯
 */
```

2.5. 显示套餐余量

使用 instanceof 判断套餐类型之后调用对应数据

格式化输出使用 Java 的 DecimalFormat 类

```
public void showRemainDetail(String number) {
    DecimalFormat formatData = new DecimalFormat( pattern: "#.0");

    System.out.print("******套餐余量查询******\n" + "您的卡号是" + number + "套餐内剩余：\n");
    MobileCard c = cards.get(number);
    if (c.serPackage instanceof TalkPackage) {
        System.out.print("通话时长：" + Math.max(((TalkPackage) c.serPackage).talkTime - c.realTalkTime, 0) + "分钟\n"
                + "短信条数：" + Math.max(((TalkPackage) c.serPackage).smsCount - c.realSMSCount, 0) + "条\n");
    } else if (c.serPackage instanceof SuperPackage) {
        System.out.print( "通话时长：" + Math.max(((SuperPackage) c.serPackage).talkTime - c.realTalkTime, 0) + "分钟\n"
                + "短信条数：" + Math.max(((SuperPackage) c.serPackage).smsCount - c.realSMSCount, 0) + "条\n"
                + "上网流量：" + formatData.format(Math.max(((SuperPackage) c.serPackage).flow - c.realFlow, 0)) + "GB\n");
    } else {
        System.out.print( "上网流量：" + formatData.format(Math.max(((NetPackage) c.serPackage).flow * 1024 - c.realFlow, 0)) + "MB\n");
    }
}
```

2.6. 本月账单查询

格式化输出使用 Java 的 DecimalFormat 类

```
public void showAmountDetail(String number) {
    DecimalFormat formatData = new DecimalFormat( pattern: "#.0");

    MobileCard c = cards.get(number);

    System.out.print(
        "******本月账单查询******\n" +
        "您的卡号：" + number + "，当月账单：\n" +
        "套餐资费：" + formatData.format(c.serPackage.price) + "元\n" +
        "合计：" + formatData.format(c.consumAmount) + "元\n" +
        "账户余额：" + formatData.format(c.money) + "元\n");
}
```

2.7. 打印消费记录

为避免频繁调用 print 函数，将所有数据使用 append 添加到缓冲字符串，最后一次性输出。

向屏幕输出完成后，调用 File 类将缓冲字符串写入文件

```
public void printConsumInfo(String number) {
    StringBuilder content = new StringBuilder();
    content.append("**********").append(number).append("消费记录**********\n").append("序号\t类型\t数据[通话(分钟)/上网(MB)/短信(条)]\n");
    List<ConsumInfo> list = consumInfos.get(number);
    for (int i = 0; i < list.size(); i++) {
        content.append(i + 1).append(".\t").append(list.get(i).type).append("\t").append(list.get(i).consumData).append("\n");
    }
    System.out.print(content);

    try {
        File file = new File( pathname: number + " 消费记录.txt");
        file.createNewFile();
        FileWriter writer = new FileWriter(file);
        writer.write(content.toString());
        writer.flush();
        writer.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

2.8. 显示资费说明

资费说明.txt 被预先制作好放置于项目根目录下，程序会尝试读取并输出

```java
public void showDescription() {

    try {
        FileReader fileReader = new FileReader( fileName: "资费说明.txt");
        BufferedReader bufferedReader = new BufferedReader(fileReader);
        String line = bufferedReader.readLine();

        while (line != null) {  //循环输出每一行数据
            System.out.println(line);
            line = bufferedReader.readLine();
        }

        bufferedReader.close();
        fileReader.close();
    } catch (FileNotFoundException e) {
        System.out.println("资费说明文件未找到");
    } catch (IOException e) {
        System.out.println("文件读取错误");
    }
}
```

2.9. 更换套餐

先使用 instanceof 检查套餐是否即将更换

再检查余额是否满足新套餐月资费，最后更换套餐，扣除余额与各项使用时长并显示新套餐与账户信息。

```java
public void changingPack(String number, String packNum) {
    MobileCard c = cards.get(number);

    //检查更换套餐是否发生了变更
    ServicePackage preSp = c.serPackage;
    if ((packNum.equals("1") && preSp instanceof TalkPackage)
            || (packNum.equals("2") && preSp instanceof NetPackage)
            || (packNum.equals("3") && preSp instanceof SuperPackage)) {
        System.out.println("对不起，您已经是该套餐用户，无需更换套餐！");
        return;
    }

    //检查余额
    ServicePackage toSp;
    if (packNum.equals("1")) toSp = new TalkPackage();
    else if (packNum.equals("2")) toSp = new NetPackage();
    else toSp = new SuperPackage();

    if (c.money < toSp.price) {
        System.out.println("对不起，您的余额不足以支付新套餐本月资费，请充值后再办理更换套餐业务！");
        return;
    }

    //扣除套餐费用，清空使用时间，更换套餐
    c.money -= toSp.price;
    c.realTalkTime = c.realSMSCount = c.realFlow = 0;
    c.serPackage = toSp;
    System.out.println("套餐更换成功");

    //输出相关信息
    c.showMeg();
    c.serPackage.showInfo();
}
```

2.10. 充值

最低充值限额检查被分离到菜单方法中进行，这里不做重复检查。

充值完成后按要求的精度输出。

```java
public void chargeMoney(String number, double money) {
    //充值金额的检查在二级菜单中已完成，不做重复检查
    DecimalFormat formatData = new DecimalFormat( pattern: "#.0");    //用于控制精度
    MobileCard mobileCard = cards.get(number);
    mobileCard.money += money;
    System.out.println(String.format("充值成功，当前话费余额为%s元。", formatData.format(mobileCard.money)));
}
```

2.11. 使用嗖嗖

首先使用 random 类生成随机数，调用对应的场景，检查该卡套餐确实可以进行该场景后继续，否则重复生成随机场景。

```
public void useSoso(String number) {
    Random random = new Random();
    int i;
    MobileCard c = cards.get(number);
    Scene scene;
    ConsumInfo info;
    boolean flag = false;

    do{
        i = random.nextInt( bound: 6);
        scene = scenes.get(i);

        if(c.serPackage instanceof SuperPackage)
            flag = true;
        else if(c.serPackage instanceof TalkPackage)
            flag = scene.type.equals("通话") || scene.type.equals("短信");
        else
            flag = scene.type.equals("上网");
    }while(!flag);
```

随后分场景类型调用对应套餐的功能性方法，并通过异常捕获，检查是否出现了余额不足。并进行相应的异常处理。具体扣款与检查逻辑分离至套餐内部实现。一切完成后添加消费记录。

```
try {
    if(scene.type.equals("通话"))
        ((CallService)c.serPackage).call(scene.data, c);
    else if(scene.type.equals("短信"))
        ((SendService)c.serPackage).send(scene.data, c);
    else
        ((NetService)c.serPackage).netPlay(scene.data, c);
} catch (InsufficientBalanceException e) {
    System.out.println("本次已" + e.done() + ", 您的余额不足，请充值后再使用！");
    e.printStackTrace();
} finally {
    System.out.print(scene.description + scene.type + scene.data);
    if (scene.type.equals("通话")) System.out.println("分钟");
    else if (scene.type.equals("上网")) System.out.println("MB");
    else System.out.println("条");
}

if (consumInfos.get(number) == null) {
    System.out.print("不存在此卡的消费记录，");
}
info = new ConsumInfo(number, scene.type, scene.data);
addConsumInfo(number, info);
System.out.println("已添加一条消费记录。");
```

3. 电话卡实体类

3.1. 构造函数

为方便工具类中的方法，这里保留无参构造函数，同时也编写了有参构造函数。

3.2. 打印电话卡信息

使用 DecimalFormat 控制精度

```
public void showMeg(){
    DecimalFormat formatData = new DecimalFormat( pattern: "#.0");
    System.out.println("卡号：" + cardNumber + "  用户名：" + userName + "  当前余额" + formatData.format(money));
}
```

4. 用户信息类

4.1. 构造函数
初始化用户通话记录中各项变量

5. 场景类

5.1. 构造函数
初始化场景各项成员变量，包括消费类型，数额与详细信息

6. 异常类 InsufficientBalanceException

6.1. 异常类包括一个成员变量完成度 done，用于记录在发生余额不足异常时，已经完成的消费数量，构造函数为其 setter，成员方法 print 为其 getter。

```
class InsufficientBalanceException extends Exception {
    private String done;

    public InsufficientBalanceException(String done) { this.done = done; }

    public String done() { return done; }
}
```

7. 套餐类的公共抽象父类 ServicePackage

7.1. 成员变量 price 表示套餐月租价格，方便多态中的调用。成员方法 public void showInfo()用于打印套餐详情。

8. 三个套餐类

8.1. 各个套餐类中自带有各自有的套餐余量成员变量（talkTime、smsCount、flow）与相应的功能函数（call、send、netPlay），构造函数将会把套餐余量预置为套餐每月额度。

8.2. 每个类的三种功能函数逻辑大体一致，这里以 SurperPackage 的 call 成员方法为例展示。

8.2.1. Call 方法

该方法分多钟情况进行套餐余量检查:

1. 套餐未用完但套餐余量不足以完成本次消费，将会扣除套餐余量后合并入情况 2 中继续执行
2. 套餐已经用完
   检查余额判断是否能够完成消费需求，抛出异常或扣费返回
3. 套餐未用完且套餐余量充足
   直接扣除对应套餐余量后返回

```java
public int call(int minCount, MobileCard card) throws InsufficientBalanceException{
    if(card.realTalkTime + minCount >= talkTime){
        int temp = 0;
        if (card.realTalkTime < talkTime) {
            temp += talkTime - card.realTalkTime;
            minCount -= talkTime - card.realTalkTime;
            card.realTalkTime = talkTime;
        }
        if(card.money >= minCount * 0.2){
            card.realTalkTime += minCount;
            card.money -= minCount * 0.2;
            temp += minCount;
            return temp;
        }
        else {
            temp += (int)Math.round(card.money / 0.2);
            card.realTalkTime += temp;
            card.money = 0;
            throw new InsufficientBalanceException("通话" + temp + "分钟");
        }
    }
    else{
        card.realTalkTime += minCount;
        return minCount;
    }
}
```

9. 三个接口

9.1. 每个接口中只有对应的功能函数的声明。

```java
public interface CallService {

    /**
     * <H3>int call(int minCount, MobileCard card) throws InsufficientBalanceException</H3>
     * 通话功能实现，用于计算实际可消费额度并扣除对应套餐余量或账户余额
     * @param minCount 消费额度
     * @param card 用户电话卡实体类
     * @return int 实际消费额度
     * @throws InsufficientBalanceException 用户余额不足异常
     */

    int call(int minCount, MobileCard card) throws InsufficientBalanceException;

}
```

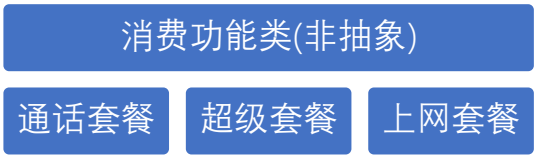
```java
public interface NetService {

    /**
     * <H3>int netPlay(int flow, MobileCard card) throws InsufficientBalanceException</H3>
     * 上网功能实现，用于计算实际可消费额度并扣除对应套餐余量或账户余额
     * @param flow 消费额度
     * @param card 用户电话卡实体类
     * @return int 实际消费额度
     * @throws InsufficientBalanceException 用户余额不足异常
     */

    int netPlay(int flow, MobileCard card) throws InsufficientBalanceException;

}
```

```java
public interface SendService {

    /**
     * <H3>int send(int count, MobileCard card) throws InsufficientBalanceException</H3>
     * 短信功能实现，用于计算实际可消费额度并扣除对应套餐余量或账户余额
     * @param count 消费额度
     * @param card 用户电话卡实体类
     * @return int 实际消费额度
     * @throws InsufficientBalanceException 用户余额不足异常
     */

    int send(int count, MobileCard card) throws InsufficientBalanceException;

}
```

反思

1. 类的结构

在按照作业要求实现三个接口的时候发现接口的实现部分代码冗余度较高，而三个类在继承其父类的时候反而不需要使用继承机制，可以通过接口来实现。所以如果想要进一步精简代码的话，可以尝试把父类 ServicePackage 替换为接口或并入父类，将三个接口合并为一个不抽象的父类并直接实现相关方法，子类中将父类方法封装并选择性调用。结构如下：



同时为了保证各个套餐不会调用自己不存在的消费方法，在子套餐类中对消费方法进行封装，工具类只能调用封装好的消费方法。