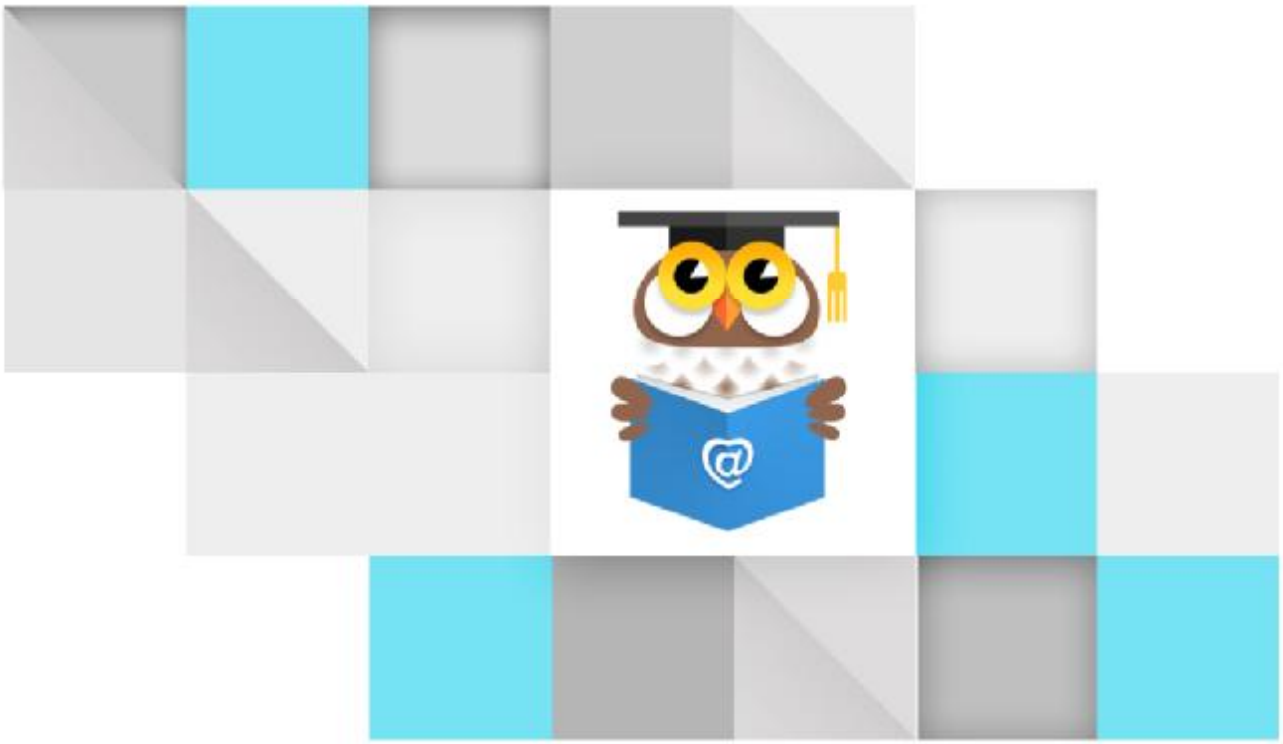
消灭泡泡糖（Java）

实训指导手册

实训场景 015 – 实现泡泡糖的积分规则



Campus Solution Group

目　录

一、任务编号：PRJ-BU2-JAVA-015

1、实训技能

- Java 面向对象编程技能

2、涉及知识点

- 数学函数与数学运算
- if 条件块
- 方法调用

3、实现效果

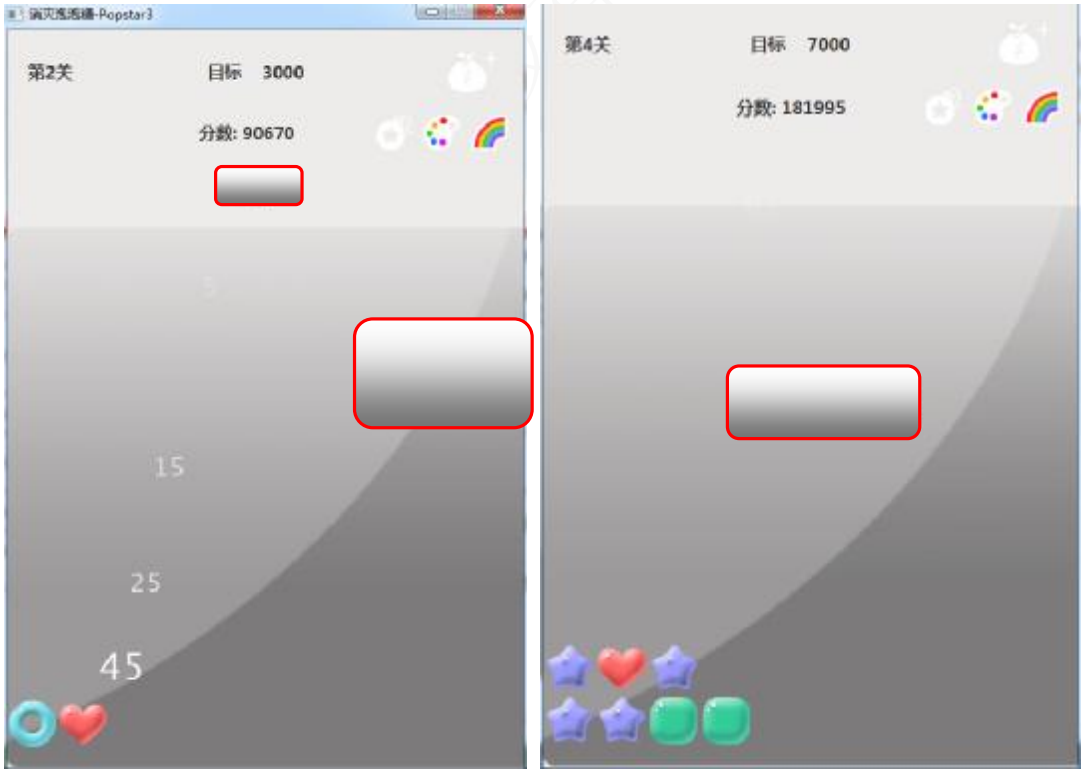


图 3-1

4、场景说明

1、业务说明：

本场景主要实现了游戏中跟分数相关的以下业务操作：

1-1. 泡泡糖消除后，系统根据被消除的泡泡糖数量计算消除得分。

1-2. 界面无可消除泡泡糖时，根据剩余泡泡糖数量结算奖励，数量越少得分越高。

1-3. 当积分达到通关目标分时，界面显示通关提示语句，且通关提示仅显示一次。

本场景的功能，均依赖于场景**PRJ-BU2-JAVA-014**中获取的配置文件得分数据。

2、实现思路：

本场景需要实现的三个功能分别对应**ScoreService**接口的三个功能函数。

2-1. 根据被消除的泡泡糖数量计算消除得分（**getScoreByStars**）：

2-1.1. 从游戏界面获取【已消除泡泡糖】数量，并获取泡泡糖数量（已实现）。

2-1.2. getScoreByStars根据泡泡糖数量，计算得分并返回游戏界面（本场景实现）。

2-1.3. 界面会将获取的分数以动画的形式展示给用户。

2-2. 根据剩余泡泡糖数量结算奖励（**getAwardScore**）：

2-2.1. 从游戏界面获取【剩余泡泡糖】数量。

2-2.2. getAwardScore根据数量，计算剩余泡泡糖的奖励分（本场景实现）。

2-2.3. 界面会将获取的分数以动画的形式展示给用户。

2-3. 界面显示通关提示语句（**isNoticePassLevel**、**isChangeLevel**）：

2-3.1. 从游戏界面获取当前游戏总得分。

2-3.2. isChangeLevel判断得分是否已经达到通关目标分，如果达到则执行切关操作。

2-3.3. isNoticePassLevel判断是否已经给出过通关提示，如果已给过则不再提示。

3、核心组件介绍：

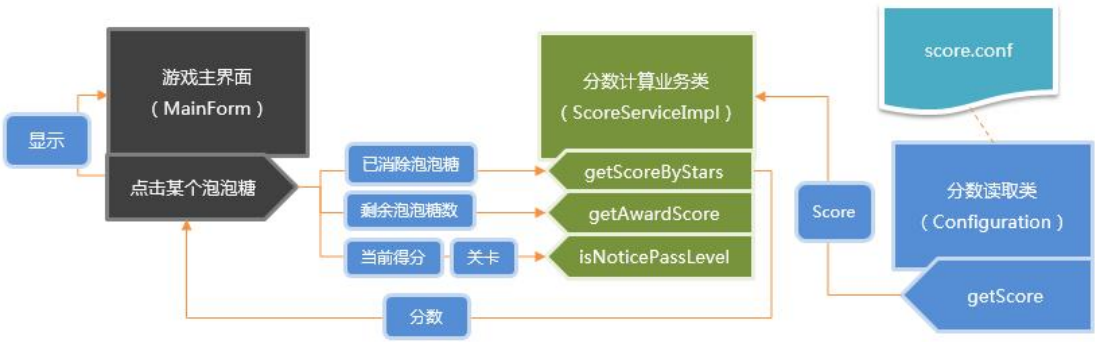


图 4-1

3-1. MainForm - 游戏界面类（本场景无需实现）：

负责游戏数据显示、响应用户在界面上的各类操作。

3-2. ScoreServiceImpl – 积分业务类

负责积分相关逻辑计算，例如：通关分数、消除得分、结算奖励等操作。

3-2.1. ScoreServiceImpl.getScoreByStars

根据被消除的泡泡糖数量计算消除得分

3-2.2. ScoreServiceImpl.getAwardScore

根据剩余泡泡糖数量结算奖励

3-2.3. ScoreServiceImpl.isNoticePassLevel

界面显示通关提示语句

3-3. Score – 记录积分信息的实体类

用于保存配置文件中的积分信息

3-4. Configuration：

该类作为得分配置文件的读取类，主要的作用是，读取配置文件中的得分信息，封装到**Score**中，以便在**ScoreServiceImpl**中进行相关业务计算。

4、了解更多：

请参考《消灭泡泡糖 - 需求说明文档》

5、前置条件：

5-1. 前置场景：**PRJ-BU2-JAVA-014** – 更新关卡通关分数

5-2. 必备知识与技能：

5-2.1. Java开发工具 （**Eclipse**）。

5-2.2. Java面向对象编程（算术运算、**if**条件块、**if**嵌套块、类的成员方法）。

5、快速开始

1、 开发环境：

1-1. Oracle JDK8.x 以上版本

1-2. Eclipse Luna（**4.4.x**）以上版本

1-3. 工程包：**PRJ_BU2_JAVA_015**

2、 进入开发环境：

详见**SPOC**平台上《**PRJ-BU2-JAVA-015** 前置任务：进入开发环境》

图 **5-1**

6、任务 1 – 获取消除奖励分数

1、任务描述：

1-1. 根据消除的泡泡糖个数，计算单次消除时所获取的奖励分值。

1-1.1. 要求：消除个数跟奖励分数成几何级数递增。

1-1.2. 算法：得分 **=** 单个泡泡糖得分 ***** 消除个数^2

1-2. 单个泡泡糖得分：本游戏固定为**5**分，存储于常量：**LOWER_SCORE**中。

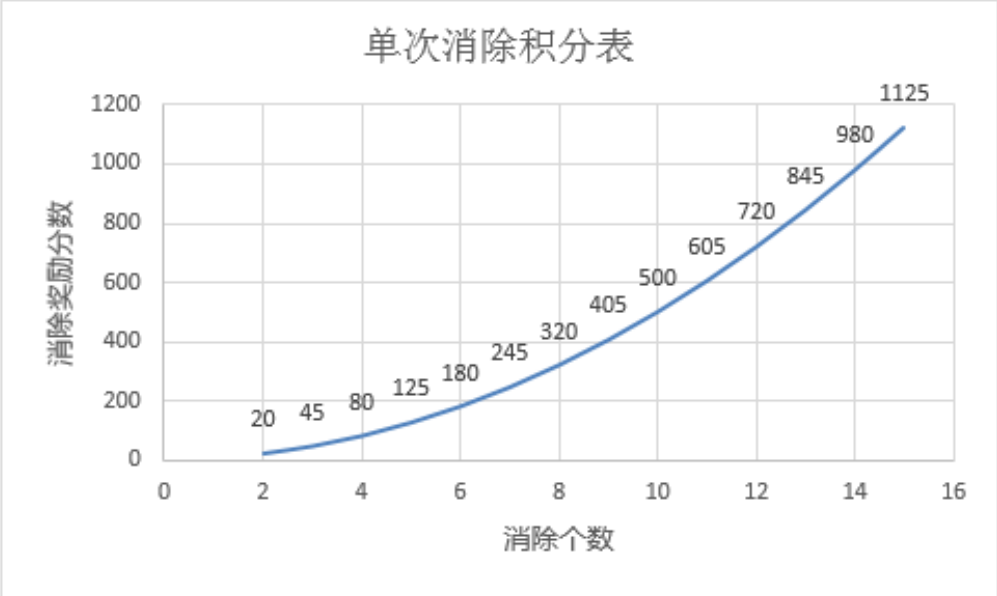


图6-1

2、推荐步骤：

2-1. 场景定位

2-1.1. 定位到：**cn.campsg.practical.bubble.service.ScoreServiceImpl**类。

2-1.2. 定位函数：**ScoreServiceImpl**类中的**getScoreByStars**方法。

2-2. 根据参数：已消除泡泡糖数量，计算获取的奖励分数。

2-2.1. 计算公式见：任务描述中的算法。

+ 提示：

1）消除的泡泡糖个数直接从方法**getScoreByStars**入参中获取。

2）单个泡泡糖得分常量**LOWER_SCORE**，在**ScoreService**接口中定义。

2-3. 返回计算所得的奖励分数。

3、验证与测试：

3-1. 运行**cn.campsg.practical.bubble.MainClass**类

3-2. 测试运行效果：

3-2.1. 点击任意可消除泡泡糖，观察是否可以显示正确的消除积分。

3-2.2. 请至少完成以下测试：

1）**2**连消：**20**分。

2）**5**连消：**125**分。

3-2.3. 结果与下图一致：

图 **6-2**

7、任务 2 – 获取结算时奖励分数

1、任务描述：

1-1. 当界面无可消除泡泡糖时，若剩余泡泡糖数量小于【限定值】，则根据剩余泡泡糖数量，进行结算奖励。

1-1.1. 要求：剩余泡泡糖数量越少奖励的分数越高，剩余泡泡糖为**0**时获得全额奖励。

1-1.2. 算法：奖励分 **=** 单个泡泡糖奖励得分 ***** （限定值 – 剩余个数)^2

1-2. 单个泡泡糖奖励得分：游戏固定为**20**分，存储于常量：**LOWER_AWARD_SCORE**中。

1-3. 限定值：奖励起算数 ,本游戏固定为**10**个(即剩余泡泡糖数小于限定值才能获得奖励)，该数值存储于常量：**AWARD_LIMIT**中。

1-4. 限定值 – 剩余个数算法是为了确保：剩余泡泡糖数量越少，则奖励的分数越高。

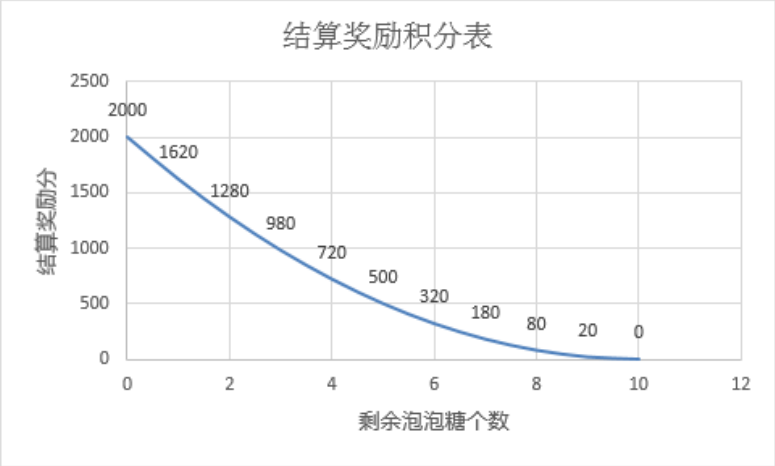


图**7-1**

2、推荐步骤：

2-1. 场景定位

2-1.1. 定位到：**cn.campsg.practical.bubble.service.ScoreServiceImpl**类。

2-1.2. 定位函数：**ScoreServiceImpl**类中**getAwardScore**方法。

2-2. 判断剩余泡泡糖数量是否满足条件：

2-2.1. 如果剩余泡泡糖数量小于【限定值】，则计算结算奖励并返回。

2-2.2. 否则，表示没有结算奖励返回**0**。

2-3. 根据参数：剩余泡泡糖数量，计算获取的奖励分数

+ 提示：

1）限定值：奖励起算数，本游戏固定为**10**个（即剩余泡泡糖数小于限定值才能获得奖励）。

2）剩余泡泡糖数量可以从方法**getAwardScore**参数中获取。

3）泡泡糖剩余数量的限定值，存储于**ScoreService**接口的**AWARD_LIMIT**常量中。

3、验证与测试：

3-1. 临界点上限测试**1**：测试消除所有泡泡糖的得分

3-1.1. 找到**cn.campsg.practical.bubble.service.StarServiceImpl**类中的**createStars**函数。

3-1.2. 将方法中调用的**testForSync**函数注释掉，这样界面就会生成**10*10**绿色泡泡糖矩阵

3-1.3. 运行项目，选择**cn.campsg.practical.bubble.MainClass**类作为入口

3-1.4. 点击任意绿色泡泡糖，观察结算奖励分是否为**2000**分，如图**7-2**

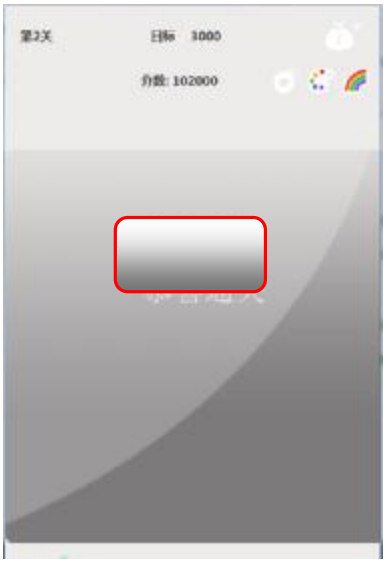


图7-2

3-2. 临界点下限测试**2**：测试剩余**10**个泡泡糖的得分

3-2.1. 找到**cn.campsg.practical.bubble.service.StarServiceImpl**类中**createStars**函数

3-2.2. 在方法**return**前，调用测试函数**test002**，入参为泡泡糖矩阵**-stars**，这样界面会生成**10**个不可消除泡泡糖。

3-2.3. 运行项目，选择**cn.campsg.practical.bubble.MainClass**类作为入口

3-2.4. 点击任意绿色泡泡糖，观察结算奖励分是否为**0**分，如图**7-3**

图 **7-3**

8、任务 3 – 显示通关提示

1、任务描述：

1-1. 当前任务主要实现：当游戏积分达到通关目标分时，在游戏界面显示通关提示语句。

1-2. 通关提示语句为：恭喜通关，保存于**Message**类中的**LEVEL_OK**常量中。

1-3. 本通关提示仅显示一次。

1-4. 为实现本任务我们需要完成以下两个判断函数：

1-4.1. 判断当前游戏积分是否达到关卡的通关目标分（**isChangeLevel**函数）。

1-4.2. 判断当前关卡的通关提示是否已经提醒过（**isNoticePassLevel**函数）。

1-4.3. 流程详见图**8-1**；

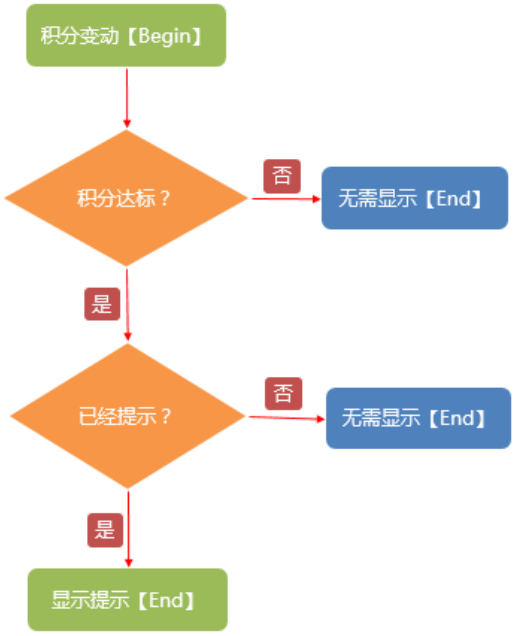


图**8-1**

2、推荐步骤：

2-1. 场景定位：**cn.campsg.practical.bubble.service.ScoreServiceImpl**类

2-2. 利用**isChangeLevel**函数，判断当前游戏积分是否达到关卡通关目标分。

2-2.1. 步骤定位：找到**ScoreServiceImpl**类的**isChangeLevel**函数

2-2.2. 获取当前关卡通关目标分数（**levelScore**）。

+ 提示：

1） 通过**Configuration**对象获取**Score**对象，并获取**levelScore**属性。

2-2.3. 判断当前游戏积分是否达到关卡通关目标分（大于等于）：

1） 达标：返回**true**。

2） 不达标：返回**false**。

2-3. 判断是否需要显示通关提示：

2-3.1. 步骤定位：找到当前类中的**isNoticePassLevel**函数。

2-3.2. 判断，如果当前积分未达到通关目标分，则表示不需要进行通关提示。

+ 提示：

1） **isNoticePassLevel**函数第二个参数表示：当前积分。

2） 是否通关，可以调用任务**2**中完成的**isChangeLevel**函数。

2-4. 判断是否已提示过通关信息：

2-4.1. 在**ScoreServiceImpl**类中创建整型成员变量，该变量用于描述游戏总通关次数，默认数值为**1**。

2-4.2. 判断游戏总通关次数是否等于当前关卡号

1） 不相等：表示已经提示过通关消息，无需再次提示，并直接返回**false**。

2） 相等：表示未提示过通关消息。

3） 以上判断的目的详见提示。

2-4.3. 如果未提示过通关消息，则我们将游戏总通关次数**+1**，为下次判断做准备。

2-4.4. 返回**true**，告知游戏界面需要提示通关消息。

+ 提示：

1） **isNoticePassLevel**函数第一个参数表示：当前关卡号。

2） 游戏总通关次数：该变量默认初始值为**1**，每次提示完通关消息后会自动累加**1**，该业务有两个目的：

2-1）为下一关的判断做准备。

2-2）虽达到通关要求，但仍存在可消除的泡泡糖时，确保不再做同类提醒。

3、验证与测试：

3-1. 定位：**cn.campsg.practical.bubble.service.StarServiceImpl**类的**createStars**函数。

3-2. 注释**test002**语句（该语句为任务**2**的测试语句）。

3-3. 去除**testForSync**语句的注释（该语句为任务**3**的测试语句）。

3-4. 运行**cn.campsg.practical.bubble.MainClass**类

3-5. 测试运行效果：

3-5.1. 点击任意黄色泡泡糖，观察是否会出现通关提示。

3-5.2. 提示后，再点击可消除泡泡糖，消除时不显示通关提示。

3-5.3. 结果与下图一致：

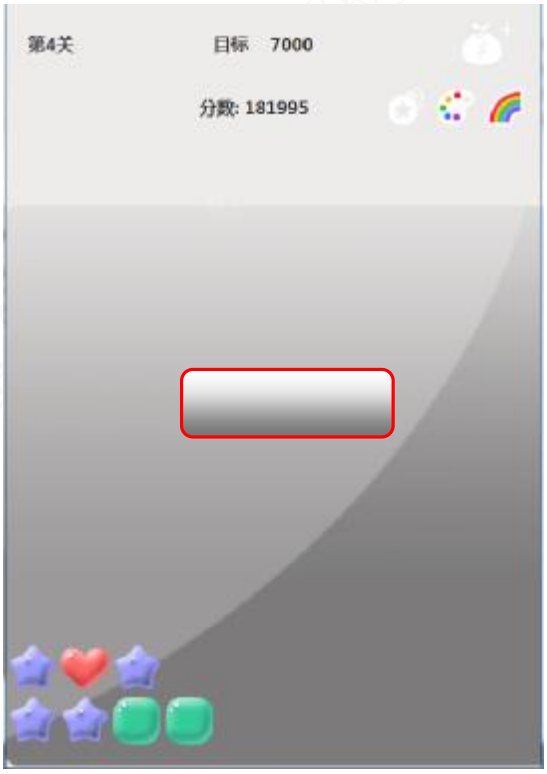


图 **8-2**

9、场景总结

Q1. 以当前案例为例，谈谈Java中按值传递数据与按引用传递数据的区别？

1. Java中八种原始数据类型都是按值传递（byte、short、int、long、float、double、boolean、char）。

2. 按值传递就是，原始数据类型利用 **"="**赋值时，变量与变量之间只通过数据交换。

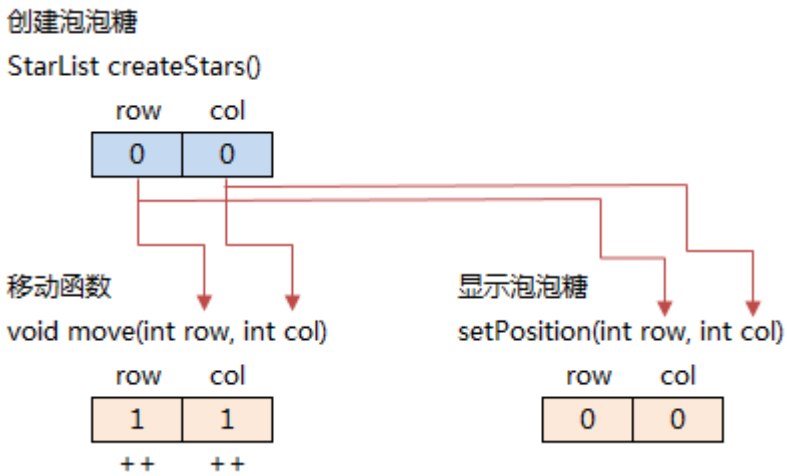
例如：

```
int row = 10;

int r = row;
```

以上语句的含义是创建一个整型变量内存空间，取名**row**并赋值**10**，再创建一个整型变量内存空间，取名**r**。将**row**空间的数据赋值给**r**空间，**row**和**r**完全是两个不同的内存空间。

3. 按值传递的场景应用效果如下图：



***** 上图一共有三个**row**变量和三个**col**变量，分别存在于**createStars**、**move**、**setPosition**中。

***** 当**createStars**调用**move**时，只是将自己的**row**变量值和**col**变量值传送给了**move**函数。

***** 虽然**move**对**row**和**col**执行了**++**操作，但是**createStars**的**row**和**col**并没有受到影响。

*** createStars**再次调用**setPosition**时，**row**和**col**仍然等于**0**，**"**泡泡糖**"**肯定不会移动。

4. Java中的三种引用数据类型都是按引用传递的（对象、接口、数组）。

5. 按值传递就是，原始引用类型利用**"="**赋值时，多个引用对象名指代的是同一个内存空间

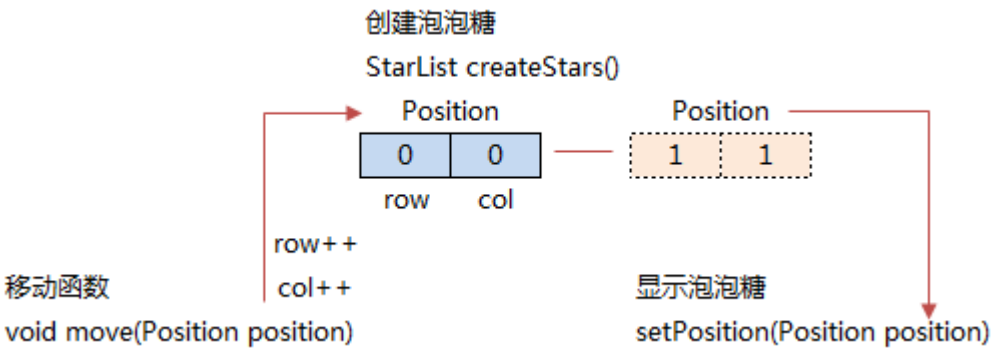
例如：

Postion p = new Position();

Postion pos = p;

以上语句的含义是：**new Position();**创建的内存空间，分别被取名为**"p"**和**"pos"**，操作**"p"**或**"pos"**其实访问的是同一块内存空间，这个与值类型差别是非常大的。

6. 按值传递的场景应用效果如下图：



***** 上图只有一个**position**对象，虽然分散在**createStars**、**move**、**setPosition**中，其实指代的是同一块内存。

【重要提示】：函数的”形式"、"实际“参数间"隐藏"了个**"="**，不理解时，可假设**"="**，再理解表达式。

【重要提示】：变量名在不同作用域中可以完全相同。