

Tarena Teaching System

Java 面向对象

学员用书



Java企业应用及互联网
高级工程师培训课程

目 录

Unit01	1
1. 对象和类	2
1.1. 面向对象程序设计	2
1.1.1. 面向过程的结构化程序设计	2
1.1.2. 什么是抽象数据类型.....	2
1.1.3. 什么是类	2
1.2. 定义一个类.....	3
1.2.1. 定义类的成员变量	3
1.2.2. 定义类的方法	4
1.3. 创建并使用对象	4
1.3.1. 使用 new 关键字创建对象.....	4
1.3.2. 引用类型变量	4
1.3.3. 访问对象的成员变量、调用方法.....	5
1.3.4. 引用类型变量的赋值.....	5
1.3.5. null 和 NullPointerException	5
经典案例	6
1. 打印员工信息	6
2. 定义 Tetris 项目中的 Cell 类	9
3. 在 Tetris 项目中实现格子的下落、左移以及获取格子的位置信息的功能	12
4. 创建 Cell 对象并添加打印方法	14
课后作业	21
Unit02	24
1. 方法	25
1.1. 方法的重载.....	25
1.1.1. 方法的签名	25
1.1.2. 方法重载及其意义	25
1.1.3. 编译时根据签名绑定调用方法	26
1.2. 构造方法	26
1.2.1. 构造方法语法结构	26
1.2.2. 通过构造方法初始化成员变量	26
1.2.3. this 关键字的使用.....	27
1.2.4. 默认的构造方法	27
1.2.5. 构造方法的重载	28
2. 数组	29
2.1. 引用类型数组	29

2.1.1. 数组是对象	29
2.1.2. 引用类型数组的声明	29
2.1.3. 引用类型数组的初始化	29
2.1.4. 数组的类型是基本类型数组	30
经典案例	32
1. 重载 drop 方法 , moveLeft 方法	32
2. 给 Cell 类添加构造方法	36
3. 给 Cell 类添加重载的构造方法	40
4. 定义 Tetris 项目中的 T 类和 J 类并测试	44
课后作业	60
Unit03	61
1. 对象内存管理	62
1.1. 对象内存管理	62
1.1.1. 对象内存管理	62
1.2. 堆内存	62
1.2.1. 对象存储在堆中	62
1.2.2. 成员变量的生命周期	62
1.2.3. 垃圾回收机制	63
1.2.4. Java 程序的内存泄露问题	63
1.2.5. System.gc()方法	63
1.3. 非堆—栈	64
1.3.1. 栈用于存放方法中的局部变量	64
1.3.2. 局部变量的生命周期	64
1.3.3. 成员变量和局部变量	64
1.4. 非堆—方法区	65
1.4.1. 方法区用于存放类的信息	65
1.4.2. 方法只有一份	65
2. 继承	65
2.1. 继承	65
2.1.1. 泛化的过程	65
2.1.2. extends 关键字	66
2.1.3. 继承中构造方法	67
2.1.4. 父类的引用指向子类的对象	67
经典案例	69
1. 构建 Tetromino 类 , 重构 T 和 J 类并测试	69
课后作业	77
Unit04	78
1. 继承	79

1.1. 重写	79
1.1.1. 方法的重写	79
1.1.2. 重写中使用 super 关键字	79
1.1.3. 重写和重载的区别	80
2. 访问控制	81
2.1. 包的概念	81
2.1.1. package 语句	81
2.1.2. import 语句	81
2.2. 访问控制修饰符	82
2.2.1. 封装的意义	82
2.2.2. public 和 private	82
2.2.3. protected 和默认访问控制	83
2.2.4. 访问控制符修饰类	83
2.2.5. 访问控制符修饰成员	83
3. static 和 final	84
3.1. static 关键字	84
3.1.1. static 修饰成员变量	84
3.1.2. static 修饰方法	84
3.1.3. static 块	85
3.2. final 关键字	85
3.2.1. final 修饰变量	85
3.2.2. final 修饰方法	86
3.2.3. final 修饰类	86
3.2.4. static final 常量	86
经典案例	87
1. 重写 T 类和 J 类的 print 方法并测试	87
课后作业	94
Unit05	98
1. 抽象类和接口	99
1.1. 使用抽象类	99
1.1.1. 抽象方法和抽象类	99
1.1.2. 抽象类不可以实例化	99
1.1.3. 继承抽象类	99
1.1.4. 抽象类的意义	100
1.2. 使用接口	100
1.2.1. 定义一个接口	100
1.2.2. 实现接口	100
1.2.3. 接口的继承	101

1.2.4. 接口和抽象类的区别.....	101
经典案例	102
1. 根据周长计算不同形状图形的面积.....	102
2. 银行卡系统(实现银联接口)	106
课后作业	115
Unit06	117
1. 抽象类和接口.....	118
1.1. 多态	118
1.1.1. 多态的意义	118
1.1.2. 向上造型	118
1.1.3. 强制转型	118
1.1.4. instanceof 关键字	119
1.2. 内部类	119
1.2.1. 定义成员内部类	119
1.2.2. 创建内部类对象	119
1.2.3. 定义匿名内部类	120
1.3. 面向对象汇总	120
1.3.1. 面向对象汇总	120
经典案例	121
1. ATM机系统	121
课后作业	130

Java 面向对象

Unit01

知识体系.....Page 2

对象和类	面向对象程序设计	面向过程的结构化程序设计
		什么是抽象数据类型
		什么是类
定义一个类		定义类的成员变量
		定义类的方法
创建并使用对象		使用 new 关键字创建对象
		引用类型变量
		访问对象的成员变量、调用方法
		引用类型变量的赋值
		null 和 NullPointerException

经典案例.....Page 6

打印员工信息	面向过程的结构化程序设计	面向过程的结构化程序设计
		什么是抽象数据类型
		什么是类
定义 Tetris 项目中的 Cell 类	定义类的成员变量	
在 Tetris 项目中实现格子的下落、左移以及获取格子的位置信息的功能	定义类的方法	
创建 Cell 对象并添加打印方法		使用 new 关键字创建对象
		引用类型变量
		访问对象的成员变量、调用方法
		引用类型变量的赋值
		null 和 NullPointerException

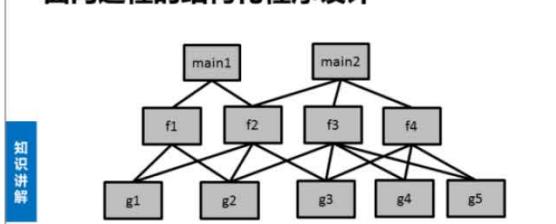
课后作业.....Page 21

1. 对象和类

1.1. 面向对象程序设计

1.1.1. 【面向对象程序设计】面向过程的结构化程序设计

面向过程的结构化程序设计



+
+

Tarena
达内科技

知识讲解

结构化程序的弊端：

1. 缺乏对数据的封装；
2. 数据和方法（对数据的操作）的分离。

1.1.2. 【面向对象程序设计】什么是抽象数据类型

什么是抽象数据类型

- 所谓抽象数据类型可以理解为：将不同类型的数据的集合组成一个整体用来描述一种新的事物；

+
+

Tarena
达内科技

知识讲解



1.1.3. 【面向对象程序设计】什么是类

什么是类

- 类定义了一种抽象数据类型。
- 类不但定义了抽象数据类型的组成（成员变量），同时还定义了可以对该类型实施的操作（方法）。

+
+

Tarena
达内科技

知识讲解

```
/** 定义雇员类 */
public class Emp{
    String name;
    int age;
    char gender;
    double salary;
}
```

在此示例中，仅仅定义了Emp类型的组成，即成员变量。该类定义了4个成员变量：String类型的name用于存放名字；int类型的age用于存放年龄；char类型的gender用于存放性别；double类型的salary用于存放工资。

Tarena
达内科技

什么是类（续1）

- 定义了Emp类以后，提升了代码的模块化以及代码的重用性，但程序依然存在问题
 - 打印信息的方法是只能针对Emp数据操作，属于Emp自身的方法，需要实现数据和方法（对该类数据的操作）的统一。

知识讲解

1.2. 定义一个类

1.2.1. 【定义一个类】 定义类的成员变量

定义类的成员变量

- 类的定义包括“成员变量”的定义和“方法”的定义，其中“成员变量”用于描述该类型对象共同的数据结构。
- Java语言中，类的成员变量的定义可以使用如下语法：

class 类名 {
 成员变量类型 变量名称;
 ...
}

+

Tarena
达内科技

定义类的成员变量（续1）

- 对象创建后，其成员变量可以按照默认的方式初始化
- 初始化对象成员变量时，其默认值的规则如下表所示：

知识讲解

成员变量的类型	默认初始值
数值类型 (byte、short、int、long、float、double)	0
boolean型	false
char型	/u0000
引用类型	null

1.2.2. 【定义一个类】 定义类的方法

Tarena
达内科技

定义类的方法

- 类中除了定义成员变量，还可以定义方法，用于描述对象的行为，封装对象的功能。
- Java语言中，可以按照如下方式定义类中的方法：

知识讲解

```
class 类名 {  
    修饰词 返回值类型 方法名称([参数列表]) {  
        方法体... ... ...  
    }  
    ... ... ...  
}
```

1.3. 创建并使用对象

1.3.1. 【创建并使用对象】使用 new 关键字创建对象

1.3.2. 【创建并使用对象】引用类型变量

Technology
Tarena
达内科技

引用类型变量

- 为了能够对实例化的对象进行访问控制，需要使用一个特殊的变量——引用。

- 引用类型变量可以存放该类对象的地址信息，通常称为“指向该类的对象”；当一个引用类型变量指向该类的对象时，就可以通过这个变量对对象实施访问。

- 除8种基本类型之外，用类、接口、数组等声明的变量都称为引用类型变量，简称“引用”。

The diagram illustrates the relationship between a class, a reference variable, and an object. On the left, a grey box labeled "类" (Class) contains the following Java code:

```
class Cell {  
    int row;  
    int col;  
}
```

To the right, a reference variable "c" is shown as a small circle containing the letter "c". An arrow labeled "引用" (Reference) points from "c" to a rectangular box labeled "对象" (Object). Inside the "对象" box are two smaller boxes labeled "row" and "col". Above the "对象" box, the text "= new Cell();" is shown, indicating the creation of a new object instance.

1.3.3. 【创建并使用对象】访问对象的成员变量、调用方法

Technology
Tarena
达内科技

访问对象的成员变量、调用方法

- 可以通过引用访问对象的成员变量或调用方法。

```
Cell c = new Cell();
```

```
c.row = 2;
```

```
c.col = 3;
```

```
c.drop();
```

```
c.moveLeft(2);
```

```
String str = c.getCellInfo();
```

知识讲解

1.3.4. 【创建并使用对象】引用类型变量的赋值

Tarena
达内科技

引用类型变量的赋值

- 引用类型变量存储的是对象的地址信息。相同类型的引用类型变量之间也可以相互赋值。
- 引用类型变量之间的赋值不会创建新的对象，但有可能会使两个以上的引用指向同一个对象。

知识讲解

```
Emp e1 = new Emp();
Emp e2 = e1; // 将e1的值(对象的地址信息)赋
              // 给e2, e2和e1指向相同的对象。
e1.name = "黄河大虾";
e2.name = "白发馍女";
System.out.println(e1.name);
```

通过引用e1和e2操作
的是相同的对象

1.3.5. 【创建并使用对象】null 和 NullPointerException

Tarena
达内科技

null和NullPointerException

- 对于引用类型变量，可以对其赋值为null。null的含义为“空”，表示还没有指向任何对象。例如：

```
Emp emp = null;  
// 引用emp中的值为null，没有指向任何对象；  
emp = new Emp();  
// 引用emp指向了一个Emp对象；
```

知识点讲解

- 当一个引用的值为null的时候，如果通过引用访问对象成员变量或者调用方法是不合逻辑的。此时，会产生NullPointerException。例如：

```
JFrame frame = null;  
frame.setSize(200,300);
```

会产生NullPointerException

经典案例

1. 打印员工信息

- 问题

构建程序，实现员工的信息的打印，打印的效果如图 - 1 所示：

The screenshot shows a Java application window titled 'Console'. The title bar also includes the text '<terminated> EmpManager [Java Application] C:\Users\jessica\AppData\Local\Temp\'. The main area of the window displays two sets of employee data separated by a dashed line. Each set consists of four lines: name, age, gender, and salary. In the second set, the salary value is higher than in the first, indicating a wage increase.

```
姓名: 白发馍女
年龄: 24
性别: 女
薪水: 6000.0
-----
姓名: 白发馍女
年龄: 24
性别: 女
薪水: 7500.0
```

图 - 1

从图-1可以看出，第二次打印员工 “” 信息时，该员工的工资上涨 25%。

- 方案

首先，一个员工包含多方面的信息数据，可以考虑使用一个对象来封装员工的数据。而多个员工的数据结构项是相同的，不同的是具体的数据，因此，可以定义一个类 Emp，用于表示员工；并在类中定义多个属性用于表示员工的各项数据。代码如下所示：

```
public class Emp {
    String name;
    int age;
    char gender;
    double salary;
}
```

然后，修改类 Emp，在类中定义打印员工信息的方法，代码如下所示：

```
public void printInfo() {
    System.out.println("-----");
    System.out.println("姓名：" + name);
    System.out.println("年龄：" + age);
    System.out.println("性别：" + gender);
    System.out.println("薪水：" + salary);
}
```

最后，在 main 方法中，创建员工对象之后，可以直接调用该对象的打印方法，以显示数据。代码如下所示：

```
Emp emp2 = new Emp();
emp2.name = "白发馍女";
emp2.age = 24;
emp2.gender = '女';
emp2.salary = 6000;

emp2.printInfo();
```

将员工信息封装为 printInfo 方法的好处为：如果员工增加其他数据项，或者打印的功能发生变化（比如打印的格式发生变化），只需要维护类 Emp 的代码即可，而调用方（方法 main）依然只需要调用相应的方法即可，从而简化了代码的维护和扩展。

• 步骤

实现此案例需要按照如下步骤进行。

步骤一：定义类 Emp

首先定义一个名为 Emp 的类，并在类中定义四个属性，分别表示员工的姓名、年龄、性别和薪资。代码如下所示：

```
public class Emp {
    String name;
    int age;
    char gender;
    double salary;
}
```

步骤二：修改类 Emp，为其定义打印员工信息的方法

修改类 Emp 中的代码，为该类定义方法，用于打印员工信息。代码如下所示：

```
public class Emp {
    String name;
    int age;
    char gender;
    double salary;

    /**
     * 打印员工信息
     */
    public void printInfo() {
        System.out.println("-----");
        System.out.println("姓名：" + name);
        System.out.println("年龄：" + age);
        System.out.println("性别：" + gender);
        System.out.println("薪水：" + salary);
    }
}
```

步骤三：构建对象，打印数据

创建 EmpManager 类，在该类的 main 方法中添加代码：定义对象，封装员工的各项数据，并调用对象的打印方法。代码如下所示：

```
public class EmpManager{  
    public static void main(String[] args) {  
  
        //创建对象  
        Emp emp2 = new Emp();  
        emp2.name = "白发馍女";  
        emp2.age = 24;  
        emp2.gender = '女';  
        emp2.salary = 6000;  
        //打印数据  
        emp2.printInfo();  
  
    }  
}
```

步骤四：修改薪资信息

将员工的薪资提高 25%，并重新打印信息。代码如下所示：

```
public class EmpManager{  
    public static void main(String[] args) {  
        //创建对象  
        Emp emp2 = new Emp();  
        emp2.name = "白发馍女";  
        emp2.age = 24;  
        emp2.gender = '女';  
        emp2.salary = 6000;  
        //打印数据  
        emp2.printInfo();  
  
        //调整薪资  
        emp2.salary *= 125.0 / 100.0;  
        emp2.printInfo();  
  
    }  
}
```

• 完整代码

本案例中，类 Emp 的完整代码如下所示：

```
public class Emp {  
    String name;  
    int age;  
    char gender;
```

```
double salary;

/**
 * 打印员工信息
 */
public void printInfo() {
    System.out.println("-----");
    System.out.println("姓名: " + name);
    System.out.println("年龄: " + age);
    System.out.println("性别: " + gender);
    System.out.println("薪水: " + salary);
}
```

类 EmpManager 的完整代码如下所示：

```
public class EmpManager{
    public static void main(String[] args) {
        //创建对象
        Emp emp2 = new Emp();
        emp2.name = "白发馍女";
        emp2.age = 24;
        emp2.gender = '女';
        emp2.salary = 6000;
        //打印数据
        emp2.printInfo();

        //调整薪资
        emp2.salary *= 125.0 / 100.0;
        emp2.printInfo();
    }
}
```

2. 定义 Tetris 项目中的 Cell 类

- 问题

Tetris 俄罗斯方块 (Tetris, 俄文：Тетрис) 是一款风靡全球的电视游戏机和掌上游戏机游戏，它由俄罗斯人阿列克谢·帕基特诺夫发明，故得此名。俄罗斯方块的基本规则是移动、旋转和摆放游戏自动输出的各种方块，使之排列成完整的一行或多行并且消除得分。由于上手简单、老少皆宜，从而家喻户晓，风靡世界。其界面效果如图 - 2 所示：

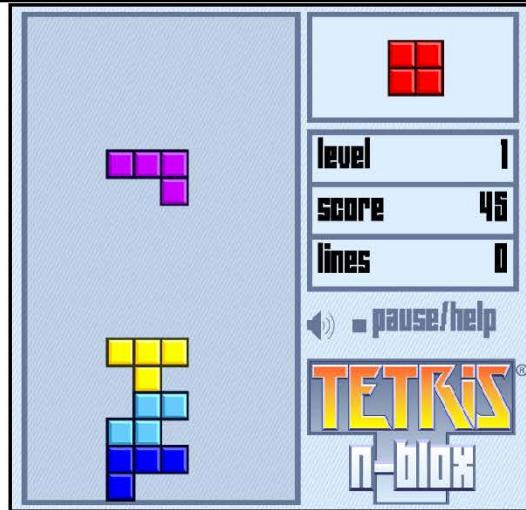


图 - 2

Tetris 游戏的基本规则为：

- 一个用于摆放正方形格子的平面虚拟场地 (Wall), 其标准大小：行宽为 10 , 列高为 20 , 以格子为单位计算 , 宽为 10 个格子 , 高为 20 个格子 ;
- 一组由 4 个小型正方形组成的规则图形 (Tetromino) , 共有 7 种 , 分别以 S 、 Z 、 L 、 J 、 I 、 O 、 T 这 7 个字母的形状来命名 , 如图 - 3 所示 :

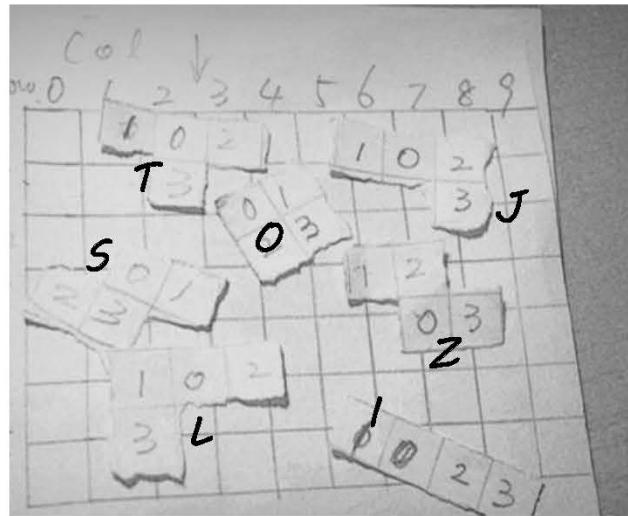


图 - 3

Tetris 游戏的逻辑为：

1. 玩家操作有 : 旋转方块 ; 以格子为单位左右移动方块 ; 让方块加速落下。
2. 方块移到区域最下方或是着地到其他方块上无法移动时 , 就会固定在该处 , 而新的方块出现在区域上方开始落下。
3. 当区域中某一行的横向格子全部由方块填满 , 则该列会消失并成为玩家的得分。同时删除的行数越多 , 得分指数上升。
4. 当固定的方块堆到区域最上方而无法消除层数时 , 则游戏结束。

-
5. 一般来说，游戏还会提示下一个要落下的方块，熟练的玩家会考虑到下一个方块，评估要如何进行。由于游戏能不断进行下去对商业用游戏不太理想，所以一般还会随着游戏的进行而加速提高难度。
 6. 预先设置的随机发生器不断地输出单个方块到场地顶部。

本案例需要定义类来表示 Tetris 游戏中的基本构造单元：格子。

• 方案

每一个格子可以由其所在行的行号和所在列的列号来确定位置。例如：图 - 4 中的白色格子，其坐标为 (3,2)，即行号为 3，列号为 2；绿色格子的坐标为 (8,7)，即行号为 8，列号为 7；黄色格子的坐标为 (17,4)，即行号为 17，列号为 4。

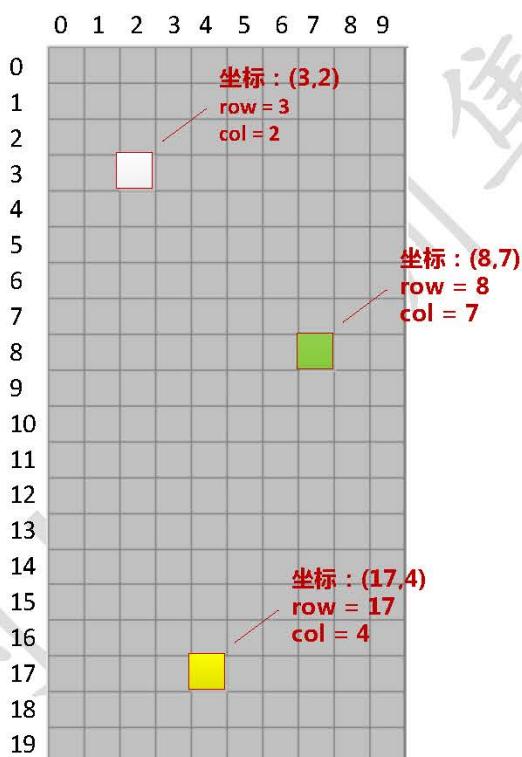


图 - 4

因此，需要定义 Cell 类来表示格子，且类中需要包含两个成员变量：row 表示行号，类型为 int；col 表示列号，类型为 int；代码如下所示：

```
public class Cell {
    int row;
    int col;
}
```

• 步骤

实现此案例需要按照如下步骤进行。

步骤一：定义 Cell 类

首先定义一个名为 Cell 的类，表示格子，并在类中添加两个 int 类型的成员变量：col 和 row，分别表示格子的横向坐标和纵向坐标。代码如下所示：

```
public class Cell {  
    int row;  
    int col;  
}
```

- **完整代码**

本案例中，类 Cell 的完整代码如下所示：

```
public class Cell {  
    int row;  
    int col;  
}
```

3. 在 Tetris 项目中实现格子的下落、左移以及获取格子的位置信息的功能

- **问题**

本案例需要实现格子下落、左移以及获取格子的位置信息的功能，详细要求如下：

1. 实现格子的下落功能，即，为 Cell 类定义下落的方法，调用该方法后，格子的位置将下降一行。
2. 实现格子的左移功能，即，为 Cell 类定义左移的方法：调用左移方法，并传入需要移动的列数，则格子将向左移动相应的列数。
3. 实现格子的 getCellInfo 方法，调用该方法，即可打印格子的位置信息。

- **方案**

要实现本案例要求的功能，解决方案如下：

1、为实现格子下落，需要为 Cell 类定义 drop 方法，并在该方法中，将格子的行数增加一行，代码如下所示：

```
public void drop() {  
    row++;  
}
```

2、为实现格子左移，需要为 Cell 类定义 moveLeft 方法，该方法需要接收一个 int 类型的参数，表示向左移动的列数，然后在该方法中，将格子的列数减少相应的数值。代码如下所示：

```
public void moveLeft(int d) {  
    col -= d;  
}
```

3、为输出格子的位置信息，可以为 Cell 类定义 getCellInfo 方法，该方法返回格子的位置信息，格式形如 row,col。代码如下所示：

```
public String getCellInfo() {  
    return row + "," + col;  
}
```

• 步骤

实现此案例需要按照如下步骤进行。

步骤一：为 Cell 类定义 drop 方法

在 Cell 类中，添加方法 drop，实现行数的增长。代码如下所示：

```
public class Cell {  
    int row;  
    int col;  
  
    //下落一行  
    public void drop() {  
        row++;  
    }  
}
```

步骤二：为 Cell 类定义 moveLeft 方法

在 Cell 类中，添加方法 moveLeft，实现列数的减少。代码如下所示：

```
public class Cell {  
    int row;  
    int col;  
  
    //下落一行  
    public void drop() {  
        row++;  
    }  
  
    //左移  
    public void moveLeft(int d) {  
        col -= d;  
    }  
}
```

步骤三：为 Cell 类定义 getCellInfo 方法

在 Cell 类中，添加方法 getCellInfo，用于返回格子的位置信息。代码如下所示：

```
public class Cell {  
    int row;  
    int col;
```

```
//下落一行
public void drop() {
    row++;
}

//左移
public void moveLeft(int d) {
    col -= d;
}

public String getCellInfo () {
    return row + "," + col;
}

}
```

- **完整代码**

本案例中，类 Cell 的完整代码如下所示：

```
public class Cell {
    int row;
    int col;

    //下落一行
    public void drop() {
        row++;
    }

    //左移
    public void moveLeft(int d) {
        col -= d;
    }

    public String getCellInfo () {
        return row + "," + col;
    }
}
```

4. 创建 Cell 对象并添加打印方法

- **问题**

之前案例已经构建了 Cell 类并定义了格子的下落、左移以及获取格子的位置信息方法。本案例要求可以设置格子的坐标，然后在界面上打印显示格子的位置信息。界面效果如图 - 5 所示：

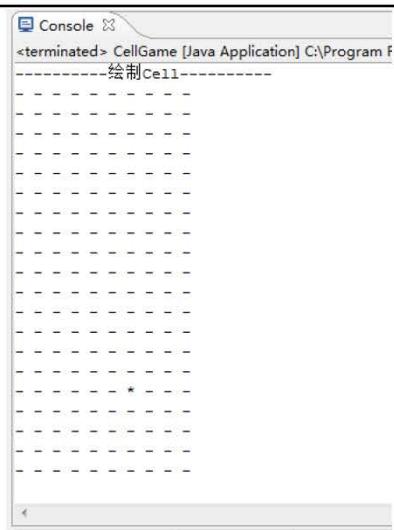


图 - 5

为了更好的理解本案例，我们用图 - 6 来标识场地以及格子的详细信息。

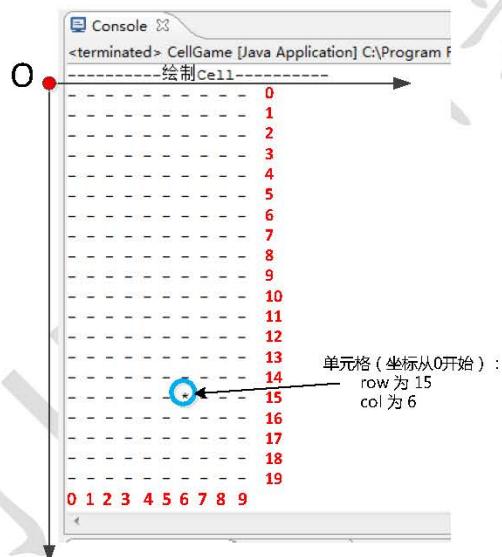


图 - 6

由图 - 6 可以看出，游戏所在的平面由 10×20 行个格子构成，可以将场地的左上角点定为原点，对行和列进行编号，列号为 $0 \sim 9$ ，行号为 $0 \sim 19$ 。

本案例详细要求如下：

1、首先需要打印出游戏所在的平面（宽 10 格，高 20 格），用 “-” 号表示平面上的每个单元；然后假设某格子的行号为 15，列号为 6，即，该格子的坐标为 $(15, 6)$ ，需要使用 “*” 号打印显示该格子，如图 - 6 中蓝色圈内所示。

2、测试格子的下落功能，界面效果如图 - 7 所示：

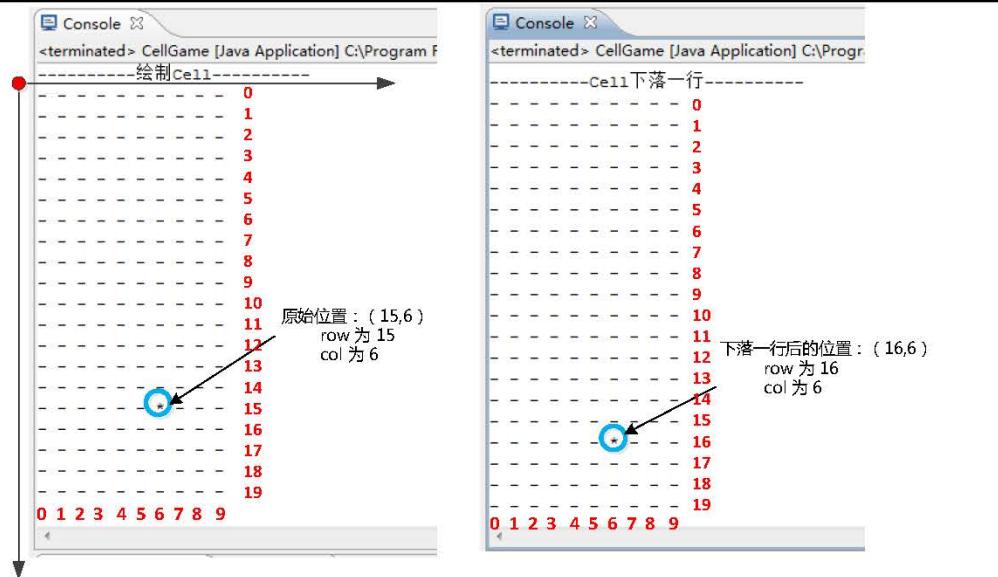


图 - 7

图 - 7 中的左图中的蓝色圈中的“*”号表示格子的原始位置，右图中蓝色圈中的“*”号表示格子下落一行后的位置。

• 方案

首先创建一个 Cell 对象，并设置行列坐标，代码如下所示：

```
Cell cell = new Cell();
cell.row = 15;
cell.col = 6;
```

然后，需要在控制台输出 10 列×20 行的场地，示意代码如下所示：

```
//总行数和总列数
int totalRow = 20;
int totalCol = 10;
for (int row = 0; row < totalRow; row++) {
    for (int col = 0; col < totalCol; col++) {
        System.out.print("- ");
    }
    System.out.println();
}
```

如果需要在场地上用“*”号表示某个单元格，则需要进行行列的判断。示意代码如下所示：

```
//总行数和总列数
int totalRow = 20;
int totalCol = 10;

//某个格子的行和列
int currentRow = 5;
```

```
int currentCol = 3;
for (int row = 0; row < totalRow; row++) {
    for (int col = 0; col < totalCol; col++) {

        if (currentRow == row && currentCol == col) {
            System.out.print("* ");//输出格子
        } else {
            System.out.print("- ");//输出场地
        }

    }
    System.out.println();
}
```

- **步骤**

实现此案例需要按照如下步骤进行。

步骤一：定义 CellGame 类

定义一个名为 CellGame 的类，并在类中添加 Java 应用程序的主方法 main，代码如下所示：

```
public class CellGame{
    public static void main(String[] args) {
    }
}
```

步骤二：创建打印 Cell 对象的方法

在 CellGame 的类中定义方法 printCell()，用于打印场地以及某个 Cell 的信息。场地用 “-” 表示，指定的格子用 “*” 表示。为标识出指定格子的信息，需要将 Cell 对象作为参数传入。代码如下所示：

```
public class CellGame{
    public static void main(String[] args) {

        public static void printCell(Cell cell) {
            int totalRow = 20;
            int totalCol = 10;

            //打印场地
            for (int row = 0; row < totalRow; row++) {
                for (int col = 0; col < totalCol; col++) {
                    if (cell.row == row && cell.col == col) {
                        //打印指定的格子
                        System.out.print("* ");
                    } else {
                        System.out.print("- ");
                    }
                }
                System.out.println();
            }
        }
    }
}
```

步骤三：创建 Cell 对象，并打印

在 main 方法中添加代码，创建一个 Cell 对象，设置其位置后，调用上一步中所创建的打印方法打印信息。代码如下所示：

```
public class CellGame{
    public static void main(String[] args) {

        System.out.println("-----绘制 Cell-----");
        //创建 Cell 对象，并打印
        Cell cell = new Cell();
        cell.row = 15;
        cell.col = 6;
        printCell(cell);

    }

    public static void printCell(Cell cell) {
        int totalRow = 20;
        int totalCol = 10;

        //打印场地
        for (int row = 0; row < totalRow; row++) {
            for (int col = 0; col < totalCol; col++) {
                if (cell.row == row && cell.col == col) {
                    //打印指定的格子
                    System.out.print("* ");
                } else {
                    System.out.print("- ");
                }
            }
            System.out.println();
        }
    }
}
```

步骤四：测试 drop 方法

在 CellGame 类的 main 方法中继续添加代码：调用 cell 对象的 drop 方法，实现格子下落一行，然后调用打印方法打印信息。代码如下所示：

```
public class CellGame{
    public static void main(String[] args) {
        System.out.println("-----绘制 Cell-----");
        //创建 Cell 对象，并打印
        Cell cell = new Cell();
        cell.row = 15;
        cell.col = 6;
        printCell(cell);

        //调用 drop 方法，下落一行
        System.out.println("-----Cell 下落一行-----");
        cell.drop();
    }
}
```

```

    printCell(cell);

}

public static void printCell(Cell cell) {
    int totalRow = 20;
    int totalCol = 10;

    //打印场地
    for (int row = 0; row < totalRow; row++) {
        for (int col = 0; col < totalCol; col++) {
            if (cell.row == row && cell.col == col) {
                //打印指定的格子
                System.out.print("* ");
            } else {
                System.out.print("- ");
            }
        }
        System.out.println();
    }
}

```

测试 moveLeft 方法和 getCellInfo 方法与测试 drop 方法类似 , 请自行测试。

• 完整代码

本案例中 , 类 Cell 的完整代码如下所示 :

```

public class Cell {
    int row;
    int col;

    //下落一行
    public void drop() {
        row++;
    }
    //左移
    public void moveLeft(int d) {
        col -= d;
    }
    public String getCellInfo () {
        return row + "," + col;
    }
}

```

类 CellGame 的完整代码如下所示 :

```

public class CellGame{
    public static void main(String[] args) {
        System.out.println("-----绘制 Cell-----");
        //创建 Cell 对象 , 并打印
        Cell cell = new Cell();
        cell.row = 15;
        cell.col = 6;
        printCell(cell);

        //调用 drop 方法 , 下落一行
        System.out.println("-----Cell 下落一行-----");
    }
}

```

```
        cell.drop();
        printCell(cell);
    }

    public static void printCell(Cell cell) {
        int totalRow = 20;
        int totalCol = 10;

        //打印场地
        for (int row = 0; row < totalRow; row++) {
            for (int col = 0; col < totalCol; col++) {
                if (cell.row == row && cell.col == col) {
                    //打印指定的格子
                    System.out.print("* ");
                } else {
                    System.out.print("- ");
                }
            }
            System.out.println();
        }
    }
}
```

课后作业

1. 关于方法的返回值和 return 语句，下面说法错误的是：

- A. return 语句用于终止当前方法的执行
- B. 如果方法的返回类型为 void，则方法中不能出现 return 语句
- C. return 关键字还会停止方法的执行；如果方法的返回类型为 void，则可使用没有值的 return 语句来停止方法的执行
- D. 定义有返回值的方法，必须使用 return 关键字返回值，且 return 关键字的后面必须是与返回类型匹配的值

2. 请描述类和对象的关系

3. 请描述引用类型和基本类型的区别

4. 为 Cell 类添加右移的方法

本案例需要实现格子的右移功能，即需要为 Cell 类定义右移的方法。该方法需要使用重载来分别实现两种功能：

功能 1：调用右移方法，不需要传入任何参数，则格子向右移动一列，如图 - 1 上中间的图形所示；

功能 2：调用右移方法，并传入需要移动的列数，则格子将向右移动相应的列数，如图 - 1 上右边的图形所示。

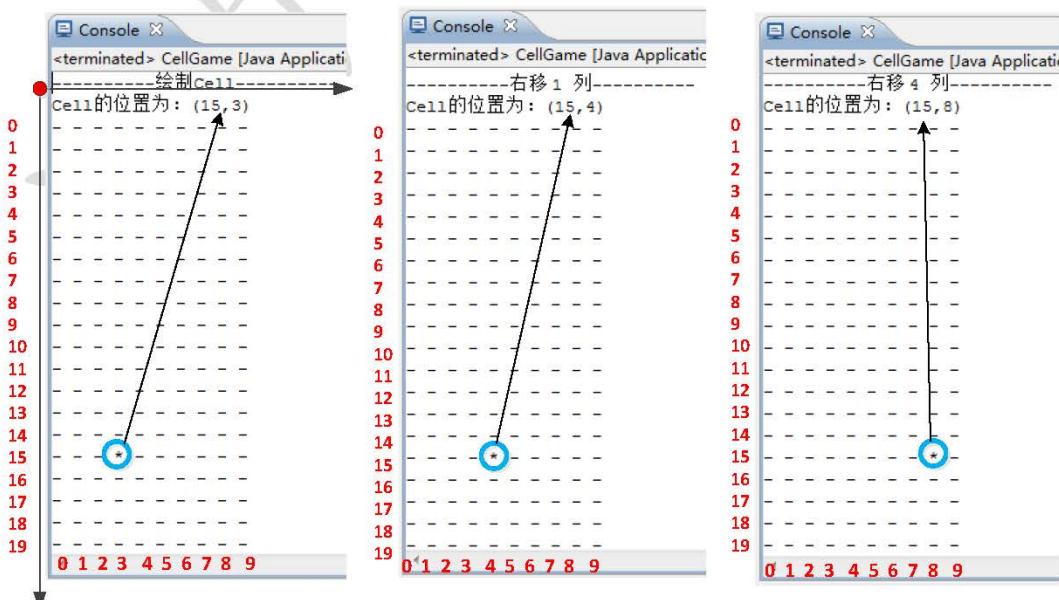


图 - 1

本案例首先需要打印出游戏所在的平面（宽 10 格，高 20 格），用“-”号表示平面上的每个单元；然后假设某格子的坐标为 (15, 3)，即，行号为 15，列号为 3，需要使用“*”号打印显示该格子，如图 - 1 中左图中的蓝色圈内所示。先调用不带参数的右移方法，则格子右移一列，如图 - 1 上中间图形上蓝色圈内所示；然后调用带参数的右移方法使得格子向右移动 4 列，并重新打印效果，如图 - 1 中右图上蓝色圈内所示。

5. 完成 CellGame (提高题 , 选作)

本案例要求完成 CellGame，用户可以在控制台上操作格子的下落、左移和右移。

游戏刚开始，将在界面上显示一个格子，界面效果如图 - 2 上左图中的蓝色圈内所示，用户可以在控制台选择输入各种操作：1 表示下落一行，2 表示左移一列，3 表示右移一列，0 表示退出。如果用户录入 1，则格子下落一行，并重新打印显示，界面效果如图 - 2 上右图中的蓝色圈内所示：

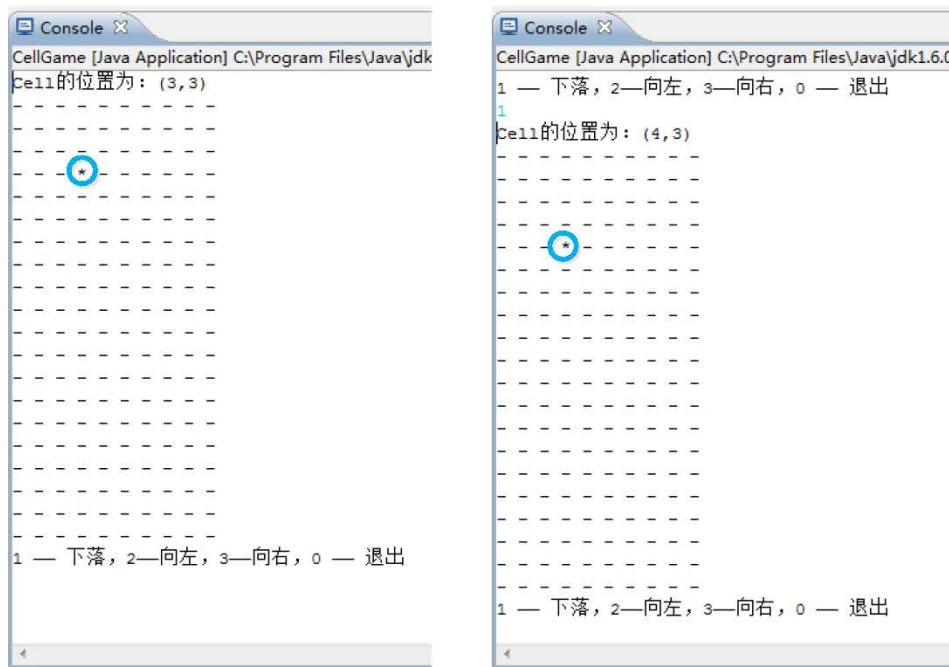


图 - 2

如果用户录入 2，则格子左移一列，并重新打印显示，界面效果如图 - 3 上左图中蓝色圈内所示；如果用户录入 3，则格子右移一列，并重新打印显示，界面如图 - 3 上右图中蓝色圈内所示：

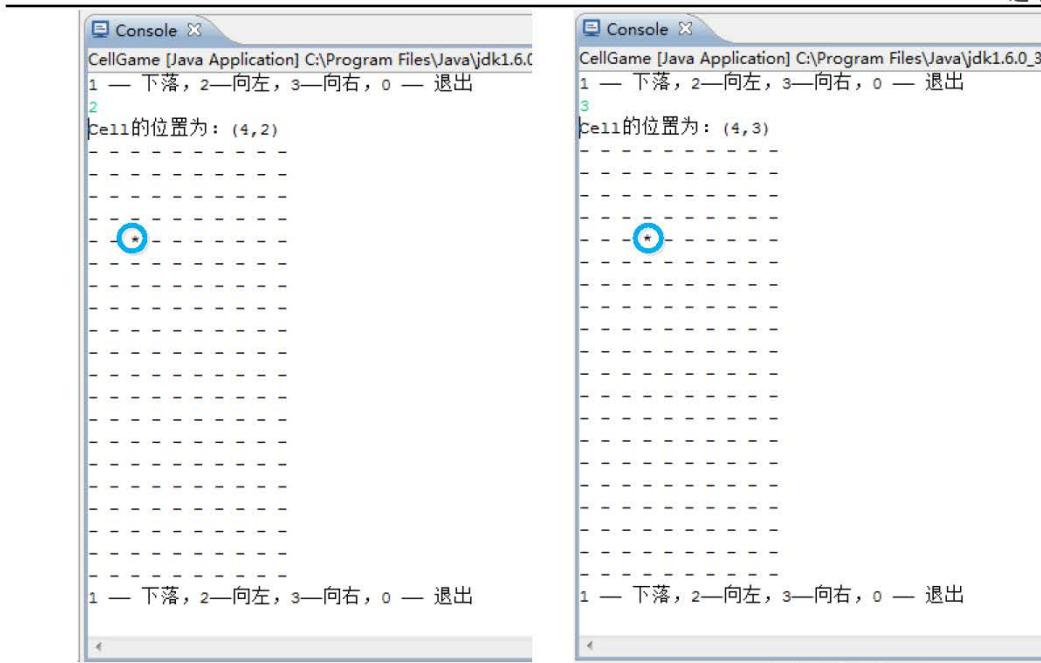
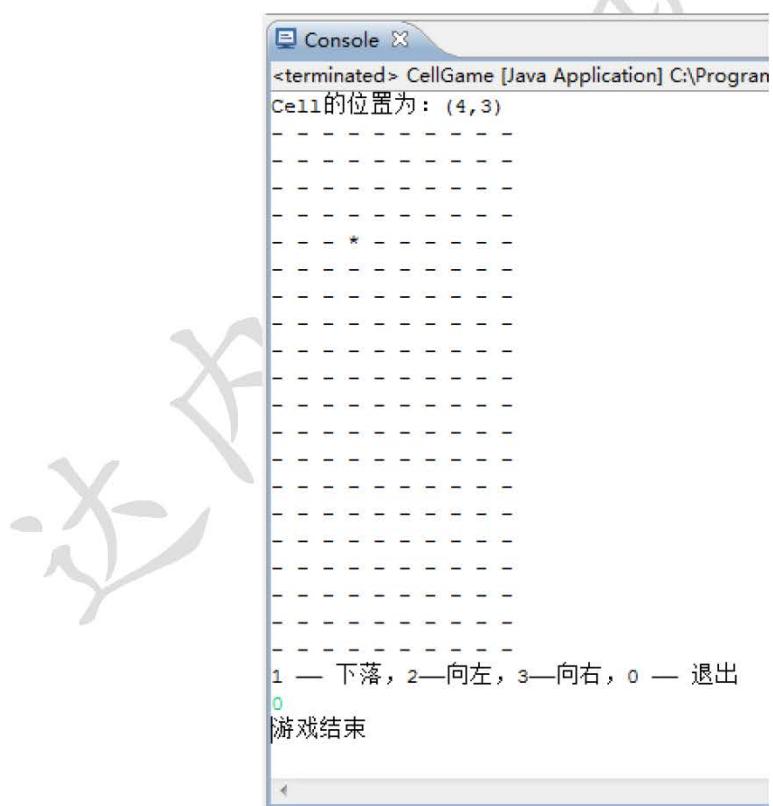


图 - 3

如果用户录入 0，则游戏结束，界面效果如图 - 4 所示：



- 4

Java 面向对象

Unit02

知识体系.....Page 25

方法	方法的重载	方法的签名
		方法重载及其意义
		编译时根据签名绑定调用方法
构造方法		构造方法语法结构
		通过构造方法初始化成员变量
		this 关键字的使用
		默认的构造方法
		构造方法的重载
数组	引用类型数组	数组是对象
		引用类型数组的声明
		引用类型数组的初始化
		数组的类型是基本类型数组

经典案例.....Page 32

重载 drop 方法 , moveLeft 方法	方法的签名
	方法重载及其意义
	编译时根据签名绑定调用方法
给 Cell 类添加构造方法	构造方法语法结构
	通过构造方法初始化成员变量
	this 关键字的使用
给 Cell 类添加重载的构造方法	默认的构造方法
	构造方法的重载
定义 Tetris 项目中的 T 类和 J 类并测试	引用类型数组的声明
	引用类型数组的初始化
	数组的类型是基本类型数组

课后作业.....Page 60

1. 方法

1.1. 方法的重载

1.1.1. 【方法的重载】方法的签名

Tarena
达内科技

方法的签名

- 方法的签名包含如下两个方面：方法名和参数列表
- 一个类中，不可以有两个方法的签名完全相同，即一个类中不可以有两个方法的方法名和参数列表都完全一样。
- 如果一个类的两个方法只是方法名相同而参数列表不同，是可以的。

知识讲解

编译错误

```
public class Cashier {  
    public boolean pay(double money) { ... }  
    public boolean pay(double money) { ... }  
}
```

编译正确

```
public class Cashier {  
    public boolean pay(double money) { ... }  
    public Boolean pay(String cardId,  
                      String cardPwd) { ... }  
}
```

1.1.2. 【方法的重载】方法重载及其意义

方法重载及其意义

假想收款窗口的设计，采用两种方式：

A. 开设三个窗口，分别用来接收现金、信用卡和支票的交付方式（分别在窗口上标明），用户根据需要选择窗口，并投入指定的物件

B. 开设一个窗口，标为“收款”，可以接收现金、信用卡和支票三种物件。该窗口可以按照输入的不同物件实施不同的操作。例如，如果输入的是现金则按现金支付，如果输入的是信用卡则按信用卡支付。



Tarena
达内科技

知识讲解

A

B

相对于A的方式，B的设计可以降低用户的负担，减少用户使用时的错误。对于用户而言，B的设计更加优雅一些。

Technology
Tarena
达内科技

方法重载及其意义（续1）

- 在Java语言中，允许多个方法的名称相同，但参数列表不同，称之为方法的重载(overload)。

```
public class PayMoney { //-----A方式
    public boolean payByCash(double money) { ... }
    public boolean payByCard(String cardId, String cardPwd) {...}
    public boolean void payByCheck
        (String compayName, double money) {...}
}
public class PayMoney { //-----B方式
    public boolean pay (double money) {...}
    public boolean pay (String cardId, String cardPwd) {...}
    public boolean pay (String compayName, double money) {...}
}
```

1.1.3. 【方法的重载】编译时根据签名绑定调用方法

编译时根据签名绑定调用方法

• 编译器在编译时会根据签名来绑定调用不同的方法，我们可以把重载的方法看成是完全不同的方法，只不过恰好方法名相同而已。

```
public boolean pay(double money) {...} // 1
public boolean pay(String cardId,
    String cardPwd) {...} // 2
public boolean pay (String compayName,
    double money) {...} // 3
pay(8888.88); // 调用方法1
pay( "12345678" , 666666 ); 调用方法2
pay( "tarena" , 8888.88); 调用方法3
```

+

1.2. 构造方法

1.2.1. 【构造方法】构造方法语法结构

构造方法语法结构

- 构造方法是在类中定义的方法，不同于其他的方法，构造方法的定义有如下两点规则：
 - 构造方法的名称必须与类名相同。
 - 构造方法没有返回值，但也不能写void。

语法：

```
[访问修饰符] 类名() {
    //构造方法体
}
```

+

1.2.2. 【构造方法】通过构造方法初始化成员变量

通过构造方法初始化成员变量

- 构造方法常用于实现对象成员变量的初始化。

```
class Cell {
    int row ; int col ;
    public Cell (int row1 , int col1 ) {
        row = row1;
        col = col1;
    }
}
class TestCell {
    public static void main(String args[]){
        Cell c1 = new Cell( 15 , 6 );
        printCell(c1);
    }
}
```

创建对象时，构造方法写在new关键字后，可以理解为：“new”创建了对象，而构造方法对该对象进行初始化。

+

1.2.3. 【构造方法】this 关键字的使用

Technology
Tarena
达内科技

this关键字的使用

- this关键字用在方法体中，用于指向调用该方法的当前对象；简单的说：哪个对象调用方法，this指的就是哪个对象。严格来讲在方法中需要通过this关键字指明当前对象。

例如：

```
public void drop () {  
    this.row ++;
```

该方法可以解释为：将调用该方法对象的成员变量row加1；
这样书写概念更加清晰。

}

- 为了方便起见，在没有歧义的情况下可以省略this：

```
public void drop () {  
    row ++;
```

}

Technology
Tarena
达内科技

this关键字的使用 (续1)

- 在构造方法中，用来初始化成员变量的参数一般和成员变量取相同的名字，这样会有利于代码的可读性，但此处就必须通过this关键字来区分成员变量和参数了（不能省略this），例如：

```
public Cell(int row, int col) {  
    this.row = row;  
    this.col = col;  
}
```

知识讲解

1.2.4. 【构造方法】默认的构造方法

Tarena
达内科技

默认的构造方法

- 任何一个类都必须含有构造方法；
- 如果源程序中没有定义，编译器在编译时将为其添加一个无参的空构造方法（称之为“默认的构造方法”）。
例如：先前的Cell类源文件中没有构造方法，编译时将为其添加如下的构造方法：Cell() { }
- 当定义了构造方法后，Java编译器将不再添加默认的构造方法。

知识讲解

默认的构造方法（续1）

```
class Cell{
    int row;  int col;
    Cell (int row , int col){
        this.row = row;
        this.col = col;
    }
}
public class CellGame {
    public static void main(String[] args) {
        Cell cell = new Cell();
    }
}
```

编译错误，当定义了构造方法Cell(int,int)后，编译器就不会提供默认的构造方法了。

知识讲解



1.2.5. 【构造方法】构造方法的重载

构造方法的重载

- 为了使用方便，可以对一个类定义多个构造方法，这些构造方法都有相同的名称（类名），方法的参数不同，称之为构造方法的重载。

```
Cell c1 = new Cell ( 5 , 6 ) → Cell ( int row , int col ) {...}
Cell c1 = new Cell ( ) → Cell () {...}
Cell c1 = new Cell ( 5 ) → Cell (int n ) {...}
```

知识讲解



构造方法的重载（续1）

- 一个构造方法可以通过this关键字调用另外一个重载的构造方法：

```
public Cell ( int row , int col ){
    this.row = row;
    this.col = col;
}

public Cell (int n) {
    this(n, n);
} 调用Cell(int, int)

public Cell0 {
    this(0,0);
}
```

知识讲解



2. 数组

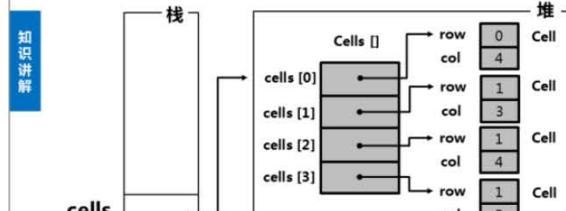
2.1. 引用类型数组

2.1.1. 【引用类型数组】数组是对象

<span style="position: absolute; left: -10px; top: 50%; width: 10px; height: 10px; background-color: #0070C0; border

引用类型数组的初始化（续1）

```
Cell[] cells = new Cell[] { new Cell(0,4), new Cell(1,3),
                           new Cell(1,4), new Cell(1,5) };
```

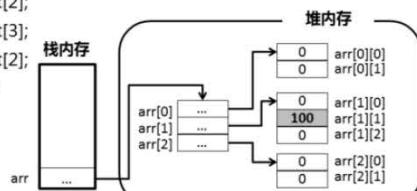


2.1.4. 【引用类型数组】数组的类型是基本类型数组

数组的类型是基本类型数组

- 数组的元素可以为任意类型，也包括数组类型

```
int [][] arr = new int[3][];
arr[0] = new int[2];
arr[1] = new int[3];
arr[2] = new int[2];
arr[1][1] = 100;
```



arr指向一个数组，该数组有三个元素，每个元素都是int类型数组，长度分别为2,3,2。



数组的类型是基本类型数组（续1）

```
int row = 3, col = 4;
int[][] arr = new int[row][col];
for (int i = 0; i < row; i++) {
    for (int j = 0; j < col; j++) {
        arr[i][j] = 0;
    }
}
```



Technology Tarena
达内科技

数组的类型是基本类型数组（续2）

- （接上）这样的数组可以用来表示类似“矩阵”这样的数据结构。`arr[i][j]`可以认为访问行号为`i`，列号为`j`的那个元素。在其他的一些语言中有专门表示这样结构的所谓二维数组；而严格的讲，Java语言中没有真正的二维数组。

知识讲解

		j=0	j=1	j=2	j=3
		arr[0][0] 0	arr[0][1] 0	arr[0][2] 0	arr[0][3] 0
		arr[1][0] 0	arr[1][1] 1	arr[1][2] 2	arr[1][3] 3
i=0	i=1	arr[2][0] 0	arr[2][1] 2	arr[2][2] 4	arr[2][3] 6
i=1	i=2				

经典案例

1. 重载 drop 方法 , moveLeft 方法

- 问题

前面的案例中，已经实现了格子下落一行以及左移多列的功能，本案例需要扩展下落和左移功能，详细要求如下：

- 调用 moveLeft 方法，不用传入参数，格子即左移一列，效果如图 - 1 中的中图所示；
- 调用 drop 方法，传入下落的行数后，格子可以下落多行，效果如图 - 1 中的右图所示。

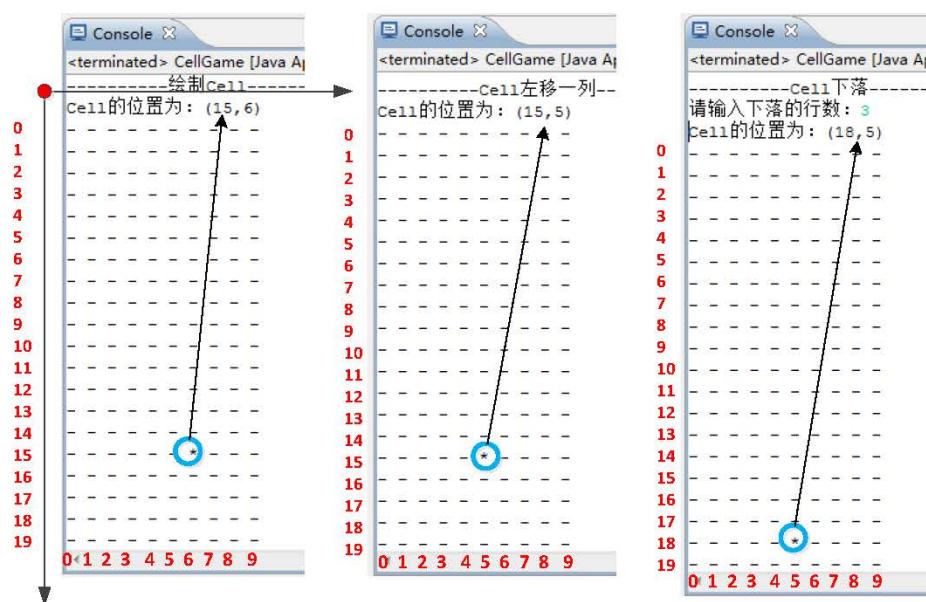


图 - 1

- 方案

Cell 类已经定义了不带参数的 drop 方法，实现下落一行的功能。如果需要能下落多列，则需要实现该方法的重载：定义带有参数的 drop 方法，代码如下所示：

```
public void drop(int d) {
    row += d;
}
```

Cell 类中已经定义的 moveLeft 方法，带有 int 类型的参数，实现在左移多列的功能。现需要调用该方法时不用传递参数，也能左移一列，则需要实现该方法的重载：定义不带参数的 moveLeft 方法。代码如下所示：

```
public void moveLeft() {  
    col--;  
}
```

- **步骤**

实现此案例需要按照如下步骤进行。

步骤一：为 Cell 类定义重载的 drop 方法

在 Cell 类中，重载 drop 方法，定义带有参数的 drop 方法，实现下落多列的功能。

代码如下所示：

```
public class Cell {  
    int row;  
    int col;  
  
    //下落一行  
    public void drop() {  
        row++;  
    }  
    //左移  
    public void moveLeft(int d) {  
        col -= d;  
    }  
    public String getCellInfo() {  
        return row + "," + col;  
    }  
  
    //重载的 drop 方法  
    public void drop(int d) {  
        row += d;  
    }  
}
```

步骤二：为 Cell 类定义重载的 moveLeft 方法

在 Cell 类中，重载 moveLeft 方法，定义不带参数的 moveLeft 方法，实现左移一列的功能。代码如下所示：

```
public class Cell {  
    int row;  
    int col;  
  
    //下落一行  
    public void drop() {  
        row++;  
    }  
    //左移  
    public void moveLeft(int d) {  
        col -= d;  
    }  
    public String getCellInfo() {  
        return row + "," + col;  
    }  
    //重载的 drop 方法
```

```

public void drop(int d) {
    row += d;
}

//重载的 moveLeft 方法
public void moveLeft() {
    col--;
}
}

```

步骤三：测试

在 CellGame 类的 main 方法中添加代码：创建坐标为 (15,6) 的格子，打印信息；然后调用 cell 对象的 moveLeft 方法，实现格子左移一列；并调用 Cell 对象的 drop 方法，实现格子下落一定的行数，并打印信息。代码如下所示：

```

import java.util.Scanner;

public class CellGame{
public static void main(String[] args) {

    System.out.println("-----绘制 Cell-----");
    //创建 Cell 对象，并打印
    Cell cell = new Cell();
    cell.row = 15;
    cell.col = 6;
    printCell(cell);

    //左移
    System.out.println("-----Cell 左移一列-----");
    cell.moveLeft();
    printCell(cell);

    //下落
    System.out.println("-----Cell 下落-----");
    System.out.print("请输入下落的行数 : ");
    Scanner scan = new Scanner(System.in);
    int rows = scan.nextInt();
    scan.close();
    cell.drop(rows);
    printCell(cell);

}

public static void printCell(Cell cell) {
    int totalRow = 20;
    int totalCol = 10;
    // 打印格子的位置信息
    System.out.println("Cell 的位置为 : (" + cell.getCellInfo() + ")");
    //打印场地
    for (int row = 0; row < totalRow; row++) {
        for (int col = 0; col < totalCol; col++) {
            if (cell.row == row && cell.col == col) {
                //打印指定的格子
            }
        }
    }
}
}

```

```
        System.out.print("* ");
    } else {
        System.out.print("- ");
    }
}
System.out.println();
}
}
```

注意：此步骤中，需要导入 `java.util` 包下的 `Scanner` 类。

- **完整代码**

本案例中，类 `Cell` 的完整代码如下所示：

```
public class Cell {
    int row;
    int col;

    //下落一行
    public void drop() {
        row++;
    }

    //左移
    public void moveLeft(int d) {
        col -= d;
    }

    public String getCellInfo() {
        return row + "," + col;
    }

    //重载的 drop 方法
    public void drop(int d) {
        row += d;
    }
    //重载的 moveLeft 方法
    public void moveLeft() {
        col--;
    }
}
```

类 `CellGame` 的完整代码如下所示：

```
import java.util.Scanner;

public class CellGame{
    public static void main(String[] args) {
        System.out.println("-----绘制 Cell-----");
        //创建 Cell 对象，并打印
        Cell cell = new Cell();
        cell.row = 15;
        cell.col = 6;
        printCell(cell);

        //左移
        System.out.println("-----Cell 左移一列-----");
        cell.moveLeft();
        printCell(cell);
    }

    //输出 Cell 对象
    public static void printCell(Cell cell) {
        System.out.print("第 " + cell.row + " 行，");
        System.out.print("第 " + cell.col + " 列");
    }
}
```

```
//下落
System.out.println("-----Cell 下落-----");
System.out.print("请输入下落的行数: ");
Scanner scan = new Scanner(System.in);
int rows = scan.nextInt();
scan.close();
cell.drop(rows);
printCell(cell);
}

public static void printCell(Cell cell) {
    int totalRow = 20;
    int totalCol = 10;
    // 打印格子的位置信息
    System.out.println("Cell 的位置为: (" + cell.getCellInfo() + ")");
    //打印场地
    for (int row = 0; row < totalRow; row++) {
        for (int col = 0; col < totalCol; col++) {
            if (cell.row == row && cell.col == col) {
                //打印指定的格子
                System.out.print("* ");
            } else {
                System.out.print("- ");
            }
        }
        System.out.println();
    }
}
```

2. 给 Cell 类添加构造方法

- 问题

为 Cell 类定义有参构造方法，并在构造方法中初始化 Cell 的行和列；然后创建一个坐标为 (0,4) 的格子，并打印信息，效果如图 - 2 所示：

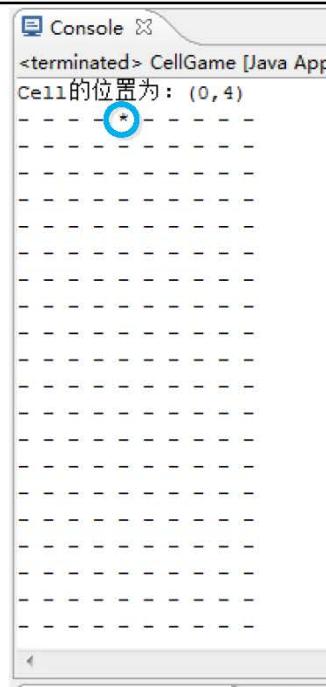


图 - 2

图 - 2 中蓝色圈中的“*”号表示所创建的格子。

• 方案

在方法中可以通过 `this` 关键字表示“调用该方法的那个对象”，因此，可以使用 `this` 关键字指向类中的成员变量，代码如下所示：

```
/**
 * 使用 this 关键字重构
 * @param row:行
 * @param col:列
 */
public Cell(int row, int col) {
    this.row = row;
    this.col = col;
}
```

• 步骤

实现此案例需要按照如下步骤进行。

步骤一：给 Cell 类添加构造方法

在 `Cell` 类中，添加带两个参数的构造方法，代码如下所示：

```
public class Cell {
    int row;
    int col;
```

```

    /**
     * 使用 this 关键字重构
     * @param row : 行
     * @param col : 列
     */
    public Cell(int row, int col) {
        this.row = row;
        this.col = col;
    }

    //下落一行
    public void drop() {
        row++;
    }

    //左移
    public void moveLeft(int d) {
        col -= d;
    }

    public String getCellInfo() {
        return row + "," + col;
    }

    //重载的 drop 方法
    public void drop(int d) {
        row += d;
    }

    //重载的 moveLeft 方法
    public void moveLeft() {
        col--;
    }
}

```

步骤二：测试有参构造方法

在 CellGame 类的 main 方法中添加代码：使用构造方法创建坐标为 (0, 4) 的格子，并打印信息，代码如下所示：

```

public class CellGame{
    public static void main(String[] args) {

        System.out.println("-----绘制 Cell-----");
        //创建 Cell 对象，并打印
        Cell cell2 = new Cell(0, 4);
        printCell(cell2);

    }

    public static void printCell(Cell cell) {
        int totalRow = 20;
        int totalCol = 10;
        // 打印格子的位置信息
        System.out.println("Cell 的位置为：" + cell.getCellInfo());
        // 打印场地
        for (int row = 0; row < totalRow; row++) {
    }
}

```

```

        for (int col = 0; col < totalCol; col++) {
            if (cell.row == row && cell.col == col) {
                //打印指定的格子
                System.out.print("* ");
            } else {
                System.out.print("- ");
            }
        }
        System.out.println();
    }
}

```

- **完整代码**

本案例中，类 Cell 的完整代码如下所示：

```

public class Cell {
    int row;
    int col;

    /**
     * 使用 this 关键字重构
     * @param row: 行
     * @param col: 列
     */
    public Cell(int row, int col) {
        this.row = row;
        this.col = col;
    }

    //下落一行
    public void drop() {
        row++;
    }

    //左移
    public void moveLeft(int d) {
        col -= d;
    }

    public String getCellInfo() {
        return row + "," + col;
    }

    //重载的 drop 方法
    public void drop(int d) {
        row += d;
    }
    //重载的 moveLeft 方法
    public void moveLeft() {
        col--;
    }
}

```

类 CellGame 的完整代码如下所示：

```

public class CellGame{
    public static void main(String[] args) {
        System.out.println("-----绘制 Cell-----");
    }
}

```

```
//创建 Cell 对象，并打印
//Cell cell2 = new Cell(0, 4);
//printCell(cell2);
}

public static void printCell(Cell cell) {
    int totalRow = 20;
    int totalCol = 10;
    // 打印格子的位置信息
    System.out.println("Cell 的位置为: (" + cell.getCellInfo() + ")");
    //打印场地
    for (int row = 0; row < totalRow; row++) {
        for (int col = 0; col < totalCol; col++) {
            if (cell.row == row && cell.col == col) {
                //打印指定的格子
                System.out.print("* ");
            } else {
                System.out.print("- ");
            }
        }
        System.out.println();
    }
}
```

3. 给 Cell 类添加重载的构造方法

- 问题

给 Cell 类添加重载的构造方法，详细要求如下：

1. 为 Cell 类定义默认构造方法，使用该构造方法创建对象，并打印显示。
2. 继续为 Cell 类定义构造方法，要求该构造方法接收一个 Cell 类型的参数。使用该构造方法，创建一个位置为 (0,4) 的格子，并打印显示。

- 方案

可以对一个类定义多个构造方法，这些构造方法都有相同的名称（类名），但是参数不同，称之为构造方法的重载。在创建对象时，Java 编译器会根据不同的参数调用不同的构造方法。

我们已经为 Cell 类定义了带有两个参数的构造方法，这样，就可以使用如下代码创建 Cell 对象：

```
Cell cell2 = new Cell(0, 4); //row 为 0, col 为 4
```

但是，此时，就不能再用如下的方式创建 Cell 对象了：

```
Cell cell2 = new Cell();
```

这是因为，当我们为类定义了构造方法后，Java 编译器将不再为该类添加默认的构造方法。如果依然希望能够使用默认的方式来创建对象，则需要为类重新定义默认的构造方法，

代码如下所示：

```
public Cell() {  
}
```

在默认构造方法中，依然需要为成员变量 col 和 row 赋值，则可以使用如下代码：

```
public Cell() {  
    this(0, 0);  
}
```

我们可以继续为 Cell 类定义带有其他类型参数的构造方法，代码如下所示：

```
public Cell(Cell cell) {  
    this(cell.row, cell.col);  
}
```

然后，可以使用如下的方式创建 Cell 对象：

```
Cell cell1 = new Cell(0, 4);  
Cell cell2 = new Cell(cell1);
```

当我们为类定义了多个构造方法后，在实际使用时，可以根据实际情况选择合适的构造方法。

- **步骤**

实现此案例需要按照如下步骤进行。

步骤一：为 Cell 类添加默认构造方法

在 Cell 类中，定义默认构造方法。代码如下所示：

```
public class Cell {  
    int row;  
    int col;  
  
    /**  
     * 使用 this 关键字重构  
     * @param row:行  
     * @param col:列  
     */  
    public Cell(int row, int col) {  
        this.row = row;  
        this.col = col;  
    }  
  
    /**  
     * 默认构造方法  
     */  
    public Cell() {  
        this(0, 0);  
    }  
}
```

```
//其他方法的代码  
//...  
}
```

步骤二：测试默认构造方法

修改 CellGame 类的 main 方法，使用默认构造方法创建 Cell 对象，并进行打印测试，代码如下所示：

```
public static void main(String[] args) {  
    System.out.println("-----绘制 Cell-----");  
  
    //创建 Cell 对象，并打印  
    Cell cell2 = new Cell();  
  
    printCell(cell2);  
}
```

步骤三：再为 Cell 类添加一个构造方法

在 Cell 类中，定义接收 Cell 类型参数的构造方法。代码如下所示：

```
public class Cell {  
    int row;  
    int col;  
  
    /**  
     * 使用 this 关键字重构  
     * @param row：行  
     * @param col：列  
     */  
    public Cell(int row, int col) {  
        this.row = row;  
        this.col = col;  
    }  
  
    /**  
     * 默认构造方法  
     */  
    public Cell() {  
        this(0, 0);  
    }  
  
    /**  
     * 构造方法的重载  
     * @param cell  
     */  
    public Cell(Cell cell) {  
        this(cell.row, cell.col);  
    }  
  
    //其他方法的代码  
    //...  
}
```

步骤四：测试 Cell 类型参数的构造方法

修改 CellGame 类的 main 方法，使用新添加的构造方法创建 Cell 对象，并进行打印测试，代码如下所示：

```
public static void main(String[] args) {
    System.out.println("-----绘制 Cell-----");
    //创建 Cell 对象，并打印
    //Cell cell2 = new Cell();
    //printCell(cell2);

    //创建 Cell 对象，并打印
    Cell cell1 = new Cell(0,4);
    Cell cell2 = new Cell(cell1);
    printCell(cell2);

}
```

• 完整代码

本案例中，类 Cell 的完整代码如下所示：

```
public class Cell {
    int row;
    int col;

    /**
     * 使用 this 关键字重构
     * @param row: 行
     * @param col: 列
     */
    public Cell(int row, int col) {
        this.row = row;
        this.col = col;
    }

    /**
     * 默认构造方法
     */
    public Cell() {
        this(0, 0);
    }

    /**
     * 构造方法的重载
     * @param cell
     */
    public Cell(Cell cell) {
        this(cell.row, cell.col);
    }

    //下落一行
    public void drop() {
        row++;
    }

    //左移
    public void moveLeft(int d) {
```

```

        col -= d;
    }

    public String getCellInfo() {
        return row + "," + col;
    }

    //重载的 drop 方法
    public void drop(int d) {
        row += d;
    }

    //重载的 moveLeft 方法
    public void moveLeft() {
        col--;
    }
}

```

类 CellGame 的完整代码如下所示：

```

public class CellGame{
    public static void main(String[] args) {
        System.out.println("-----绘制 Cell-----");
        //创建 Cell 对象，并打印
        //Cell cell2 = new Cell();
        //printCell(cell2);
        //创建 Cell 对象，并打印
        Cell cell1 = new Cell(0,4);
        Cell cell2 = new Cell(cell1);
        printCell(cell2);
    }

    public static void printCell(Cell cell) {
        int totalRow = 20;
        int totalCol = 10;
        // 打印格子的位置信息
        System.out.println("Cell 的位置为: (" + cell.getCellInfo() + ")");
        //打印场地
        for (int row = 0; row < totalRow; row++) {
            for (int col = 0; col < totalCol; col++) {
                if (cell.row == row && cell.col == col) {
                    //打印指定的格子
                    System.out.print("* ");
                } else {
                    System.out.print("- ");
                }
            }
            System.out.println();
        }
    }
}

```

4. 定义 Tetris 项目中的 T 类和 J 类并测试

- 问题

在 Tetris 游戏中，游戏场地由 10 列×20 行个正方形格子构成，如图-3 所示，每个方块由四个格子组成，绘制在场地中，如图-3 所示中的红色方块。

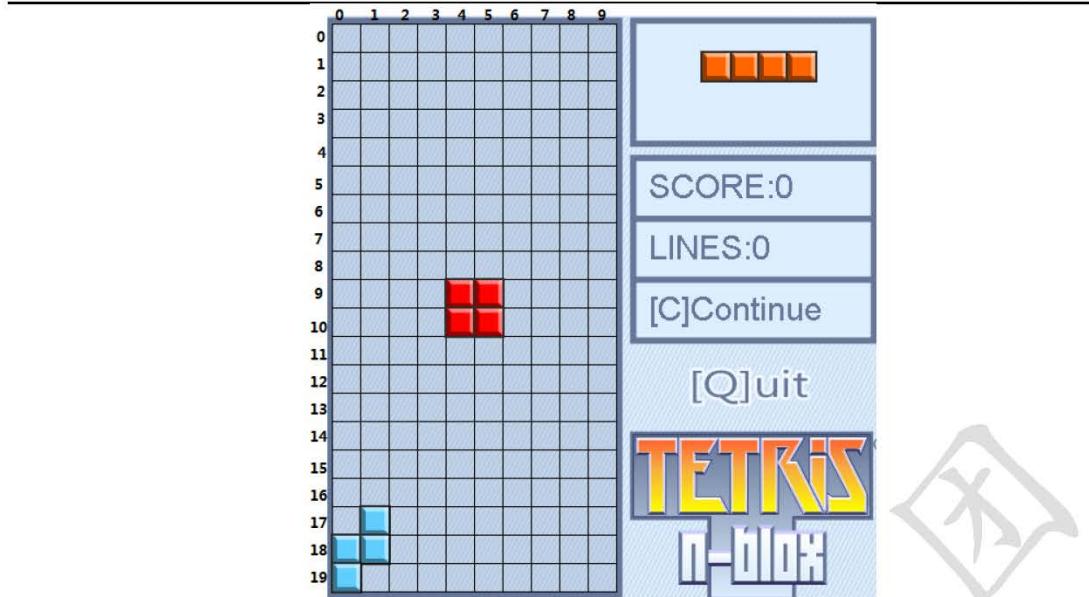


图- 3

4 个小型正方形格子组成的规则图形 (Tetromino) , 共有 7 种 , 分别以 S、Z、L、J、I、O、T 这 7 个字母的形状来命名 , 各个图形形状如图-4 所示。

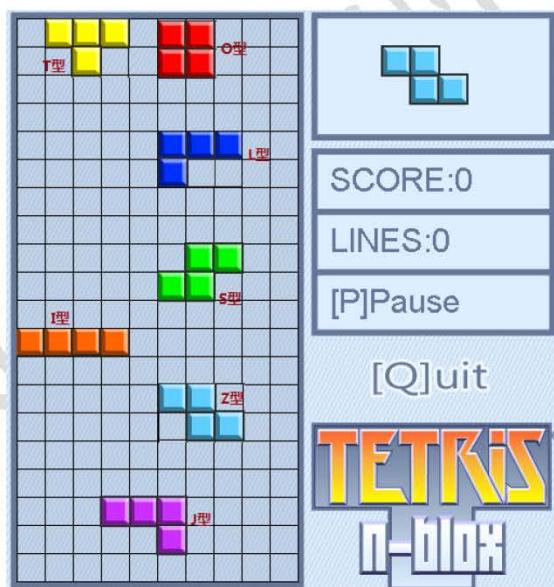


图- 4

之前案例中定义的 Cell 类 , 表示游戏场地中的一个正方形格子 , 存储其在场地中的位置。本案例中会使用到 Cell 类 , 拷贝之前案例中的该类使用即可。本案例要求定义表示 T 型方块的类 T 和 J 型方块的类 J 。 T 型方块和 J 型方块的形状如图-4 所示中黄色方块和紫色方块。

• 方案

要实现本案例要求的功能 , 解决方案如下 :

1. 定义名为 T 的类 ,由于每个方块有四个格子 ,因此 ,在 T 类中添加属性 cells , cells 属性的类型为 Cell 数组类型 ,即 , Cell[]。这样 , cells 数组中就可以存储四个格子来表示一个方块。
2. 为 T 类添加构造方法。可以提供无参数构造方法 ,及按顺时针方向、方块中第一个格子的行和列作为参数的构造方法。
3. 为了方便查看方块中四个格子的坐标 ,因此 ,定义 print 方法 ,实现按顺时针方向 ,打印方块中四个格子所在的坐标 ,并对 print 进行测试。
4. 每个方块都可以下落 ,左移和右移 ,因此 ,在 T 类中 ,定义 drop 方法实现方块下落 ;定义 moveLeft 方法实现方块左移 ;定义 moveRight 方法实现方块右移 ,并对这三个方法进行测试。
5. 实现 J 类。J 类的实现和 T 类是类似的 ,主要注意 ,在构造方法中 ,按照 J 型进行初始化 cells 属性 ,并进行测试。

• 步骤

实现此案例需要按照如下步骤进行。

步骤一：定义 T 类

首先定义一个名为 T 的类 ,代码如下所示 :

```
public class T {  
}
```

步骤二：定义属性

由于每个方块有四个格子 ,因此 ,在类 T 中定义一个名为 cells 的属性 , cells 属性的类型为 Cell 数组类型 ,即 , Cell[]。代码如下所示 :

```
public class T {  
  
    Cell[] cells;  
  
}
```

步骤三：定义构造方法

首先 ,在 T 类中添加第一个格子的行和列作为参数的构造方法 ,并在构造方法按顺时针方向初始化 T 型方块。



图- 5

从图-5中，按顺时针方向查看T型方块，可以看出，第一个格子的行列坐标是(0,1)，第二个格子的行列坐标是(0,2)，第三个格子的行列坐标是(0,3)，第四个格子的行列坐标是(1,2)。假设把第一个格子的行列坐标用(row, col)来表示，那么，第二个格子的行列坐标为(row, col+1)，第三个格子的行列坐标为(row, col+2)，第四个格子的行列坐标为(row+1, col+1)，这样，T型方块就可以用传入的参数对cells属性进行初始化了。代码如下所示：

```
public class T {
    Cell[] cells;

    /**
     * 构造方法，为属性 cells 进行初始化
     *
     * @param row 顺时针方向
     *          , 第一个坐标的行
     * @param col 顺时针方向
     *          , 第一个坐标的列
     */
    public T(int row, int col) {
        cells = new Cell[4];
        // 按顺时针方向初始化 Cell
        cells[0] = new Cell(row, col);
        cells[1] = new Cell(row, col + 1);
        cells[2] = new Cell(row, col + 2);
        cells[3] = new Cell(row + 1, col + 1);
    }
}
```

在T类中添加无参数构造方法，在该构造方法中，使用this关键字调用参数为T(int row, int col)的构造方法，并设置方块的顺时针方向的第一个格子的行和列为(0,0)，

代码如下所示：

```
public class T {  
    Cell[] cells; // 属性, 用来存储一个方块的四个格子的坐标  
  
    /**  
     * 构造方法, 为属性 cells 进行初始化  
     */  
    public T() {  
        this(0, 0);  
    }  
  
    /**  
     * 构造方法, 为属性 cells 进行初始化  
     *  
     * @param row  
     *      顺时针方向, 第一个坐标的行  
     * @param col  
     *      顺时针方向, 第一个坐标的列  
     */  
    public T(int row, int col) {  
        cells = new Cell[4];  
        // 按顺时针方向初始化 Cell  
        cells[0] = new Cell(row, col);  
        cells[1] = new Cell(row, col + 1);  
        cells[2] = new Cell(row, col + 2);  
        cells[3] = new Cell(row + 1, col + 1);  
    }  
}
```

步骤四：定义打印方法

为了方便查看方块中四个格子的坐标，因此，定义 `print` 方法。在 `print` 方法中，按顺时针方向，打印方块中四个格子所在的坐标。实现此功能，循环遍历 `cells` 数组即可。
代码如下所示：

```
public class T {  
    Cell[] cells; // 属性, 用来存储一个方块的四个格子的坐标  
  
    /**  
     * 构造方法, 为属性 cells 进行初始化  
     */  
    public T() {  
        this(0, 0);  
    }  
  
    /**  
     * 构造方法, 为属性 cells 进行初始化  
     *  
     * @param row  
     *      顺时针方向, 第一个坐标的行  
     * @param col  
     *      顺时针方向, 第一个坐标的列  
     */
```

```

/*
public T(int row, int col) {
    cells = new Cell[4];
    // 按顺时针方向初始化 Cell
    cells[0] = new Cell(row, col);
    cells[1] = new Cell(row, col + 1);
    cells[2] = new Cell(row, col + 2);
    cells[3] = new Cell(row + 1, col + 1);
}

/**
 * 按顺时针方向，打印方块中四个格子所在的坐标
 */
public void print() {
    String str = "";
    for (int i = 0; i < cells.length - 1; i++) {
        str += "(" + cells[i].getCellInfo() + "), ";
    }
    str += "(" + cells[cells.length - 1].getCellInfo() + ")";
    System.out.println(str);
}
}

```

步骤五：定义方块的下落方法

定义方块的下落的方法，即，在 T 类中，添加方块下落一个格子的方法。要实现下落一个格子，只需要循环 cells 属性，将方块中的每一个格子的行加 1 即可。代码如下所示：

```

public class T {
    Cell[] cells; // 属性, 用来存储一个方块的四个格子的坐标

    /**
     * 构造方法，为属性 cells 进行初始化
     */
    public T() {
        this(0, 0);
    }

    /**
     * 构造方法，为属性 cells 进行初始化
     *
     * @param row
     *          顺时针方向，第一个坐标的行
     * @param col
     *          顺时针方向，第一个坐标的列
     */
    public T(int row, int col) {
        cells = new Cell[4];
        // 按顺时针方向初始化 Cell
        cells[0] = new Cell(row, col);
        cells[1] = new Cell(row, col + 1);
        cells[2] = new Cell(row, col + 2);
        cells[3] = new Cell(row + 1, col + 1);
    }

    /**

```

```

    * 按顺时针方向，打印方块中四个格子所在的坐标
    */
public void print() {
    String str = "";
    for (int i = 0; i < cells.length - 1; i++) {
        str += "(" + cells[i].getCellInfo() + "), ";
    }
    str += "(" + cells[cells.length - 1].getCellInfo() + ")";
    System.out.println(str);
}

/**
 * 使方块下落一个格子
 */
public void drop() {
    for (int i = 0; i < cells.length; i++) {
        cells[i].row++;
    }
}
}

```

步骤六：定义方块的左移方法

定义方块的左移的方法，即，在 T 类中，添加方块左移一个格子的方法。要实现方块左移一个格子，只需要循环 cells 属性，将方块中的每一个格子的列减 1 即可。代码如下所示：

```

public class T {
    Cell[] cells; // 属性, 用来存储一个方块的四个格子的坐标

    /**
     * 构造方法，为属性 cells 进行初始化
     */
    public T() {
        this(0, 0);
    }

    /**
     * 构造方法，为属性 cells 进行初始化
     *
     * @param row
     *          顺时针方向，第一个坐标的行
     * @param col
     *          顺时针方向，第一个坐标的列
     */
    public T(int row, int col) {
        cells = new Cell[4];
        // 按顺时针方向初始化 Cell
        cells[0] = new Cell(row, col);
        cells[1] = new Cell(row, col + 1);
        cells[2] = new Cell(row, col + 2);
        cells[3] = new Cell(row + 1, col + 1);
    }

}

```

```

        * 按顺时针方向，打印方块中四个格子所在的坐标
    */
public void print() {
    String str = "";
    for (int i = 0; i < cells.length - 1; i++) {
        str += "(" + cells[i].getCellInfo() + "), ";
    }
    str += "(" + cells[cells.length - 1].getCellInfo() + ")";
    System.out.println(str);
}

/**
 * 使方块下落一个格子
*/
public void drop() {
    for (int i = 0; i < cells.length; i++) {
        cells[i].row++;
    }
}

/**
 * 使方块左移一个格子
*/
public void moveLeft() {
    for (int i = 0; i < cells.length; i++) {
        cells[i].col--;
    }
}

}

```

步骤七：定义方块的右移方法

定义方块的右移的方法，即，在 T 类中，添加方块右移一个格子的方法。要实现方块右移一个格子，只需要循环 cells 属性，将方块中的每一个格子的列加 1 即可。代码如下所示：

```

public class T {
    Cell[] cells;// 属性,用来存储一个方块的四个格子的坐标

    /**
     * 构造方法，为属性 cells 进行初始化
     */
    public T() {
        this(0, 0);
    }

    /**
     * 构造方法，为属性 cells 进行初始化
     *
     * @param row
     *          顺时针方向，第一个坐标的行
     * @param col
     *          顺时针方向，第一个坐标的列
     */
    public T(int row, int col) {

```

```

cells = new Cell[4];
// 按顺时针方向初始化 Cell
cells[0] = new Cell(row, col);
cells[1] = new Cell(row, col + 1);
cells[2] = new Cell(row, col + 2);
cells[3] = new Cell(row + 1, col + 1);
}

/**
 * 按顺时针方向，打印方块中四个格子所在的坐标
 */
public void print() {
    String str = "";
    for (int i = 0; i < cells.length - 1; i++) {
        str += "(" + cells[i].getCellInfo() + "), ";
    }
    str += "(" + cells[cells.length - 1].getCellInfo() + ")";
    System.out.println(str);
}

/**
 * 使方块下落一个格子
 */
public void drop() {
    for (int i = 0; i < cells.length; i++) {
        cells[i].row++;
    }
}

/**
 * 使方块左移一个格子
 */
public void moveLeft() {
    for (int i = 0; i < cells.length; i++) {
        cells[i].col--;
    }
}

/**
 * 使方块右移一个格子
 */
public void moveRight() {
    for (int i = 0; i < cells.length; i++) {
        cells[i].col++;
    }
}

```

步骤八：测试方块的右移方法

新建 TestT 类，在该类中的 main 方法中，首先，调用对象 t 的 print 方法来查看 T 型方块的原始坐标；然后，调用对象 t 的 moveRight 方法；最后，再调用对象 t 的 print 方法查看坐标的变化情况，代码如下所示：

```

public class TestT {
    public static void main(String[] args) {
        T t=new T(0,1);
    }
}

```

```

//测试 print 方法
System.out.println("原始坐标为 : ");
t.print();

// 测试 drop 方法
// t.drop();
// System.out.println("调用 drop 方法后的坐标 : ");
// t.print();

// 测试 moveLeft 方法
// t.moveLeft();
// System.out.println("调用 moveLeft 方法后的坐标 : ");
// t.print();

// 测试 moveRight 方法
t.moveRight();
System.out.println("调用 moveRight 方法后的坐标 : ");
t.print();

}
}

```

控制台的输出结果为：

```

原始坐标为 :
(0,1), (0,2), (0,3), (1,2)
调用 moveRight 方法后的坐标 :
(0,2), (0,3), (0,4), (1,3)

```

从输出结果上，可以看出 T 型方块中的每个格子的列都在原有的基础上增加了 1。界面对比效果如图-6 所示。

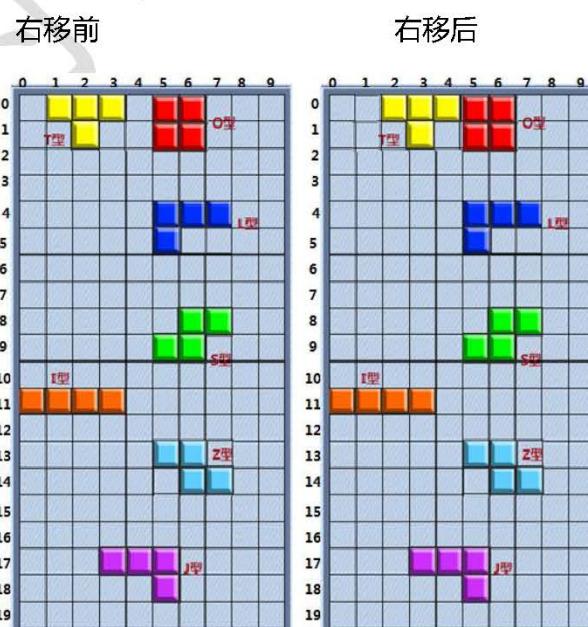


图 - 6

另外，测试下落方法，测试左移方法与测试右移方法类似，请自行测试。

步骤九：定义 J 类并实现

J 类的实现和 T 类是类似的，主要注意，在 J 类的构造方法中，按照 J 型进行初始化 cells 属性。从图-7 中，按顺时针方向查看 J 型方块，可以看出，第一个格子的行列坐标是 (17, 3)，第二个格子的行列坐标是 (17, 4)，第三个格子的行列坐标是 (17, 5)，第四个格子的行列坐标是 (18, 5)。

假设把第一个格子的行列坐标用 (row, col) 来表示，那么，第二个格子的行列坐标为 (row, col+1)，第三个格子的行列坐标为 (row, col+2)，第四个格子的坐标为 (row+1, col+2)，这样，J 型方块就可以用传入的参数对 cells 属性进行初始化了。

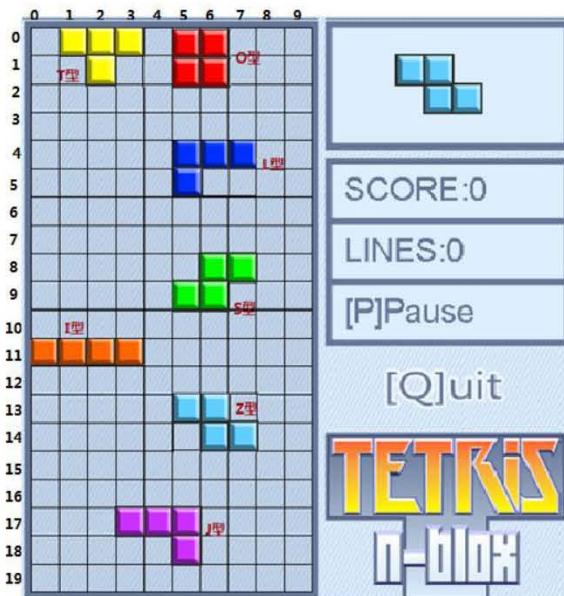


图- 7

另外，打印的方法 print、方块下落的方法 drop、方块左移的方法 moveLeft 和方块右移的方法 moveRight 与 T 类的实现方式相同。J 类代码实现如下所示：

```
public class J {
    Cell[] cells; // 属性, 用来存储一个方块的四个格子的坐标

    /**
     * 构造方法，为属性 cells 进行初始化
     */
    public J() {
        this(0, 0);
    }

    /**
     * 构造方法，为属性 cells 进行初始化
     *
     * @param row
     *      顺时针方向，第一个坐标的行
     * @param col
     *      顺时针方向，第一个坐标的列
     */
}
```

```

public J(int row, int col) {
    cells = new Cell[4];
    // 按顺时针方向初始化 Cell
    cells[0] = new Cell(row, col);
    cells[1] = new Cell(row, col + 1);
    cells[2] = new Cell(row, col + 2);
    cells[3] = new Cell(row + 1, col + 2);
}

/**
 * 按顺时针方向，打印方块中四个格子所在的坐标
 */
public void print() {
    String str = "";
    for (int i = 0; i < cells.length - 1; i++) {
        str += "(" + cells[i].getCellInfo() + "), ";
    }
    str += "(" + cells[cells.length - 1].getCellInfo() + ")";
    System.out.println(str);
}

/**
 * 使方块下落一个格子
 */
public void drop() {
    for (int i = 0; i < cells.length; i++) {
        cells[i].row++;
    }
}

/**
 * 使方块左移一个格子
 */
public void moveLeft() {
    for (int i = 0; i < cells.length; i++) {
        cells[i].col--;
    }
}

/**
 * 使方块右移一个格子
 */
public void moveRight() {
    for (int i = 0; i < cells.length; i++) {
        cells[i].col++;
    }
}

```

步骤十：测试 J 类中的方法

测试 J 类中方法的过程，与测试 T 类中方法的过程类似，这里不再赘述。TestJ 类代码如下所示：

```

public class TestJ {
    public static void main(String[] args) {
        J j=new J(17,3);
        //测试print方法
        System.out.println("原始坐标为：");
        j.print();
    }
}

```

```

    // 测试 drop 方法
    //     j.drop();
    //     System.out.println("调用 drop 方法后的坐标 : ");
    //     j.print();

    // 测试 moveLeft 方法
    //     j.moveLeft();
    //     System.out.println("调用 moveLeft 方法后的坐标 : ");
    //     j.print();

    // 测试 moveRight 方法
    j.moveRight();
    System.out.println("调用 moveRight 方法后的坐标 : ");
    j.print();
}

```

• 完整代码

本案例中，T 类的完整代码如下所示：

```

public class T {
    Cell[] cells; // 属性，用来存储一个方块的四个格子的坐标

    /**
     * 构造方法，为属性 cells 进行初始化
     */
    public T() {
        this(0, 0);
    }

    /**
     * 构造方法，为属性 cells 进行初始化
     *
     * @param row
     *         顺时针方向，第一个坐标的行
     * @param col
     *         顺时针方向，第一个坐标的列
     */
    public T(int row, int col) {
        cells = new Cell[4];
        // 按顺时针方向初始化 Cell
        cells[0] = new Cell(row, col);
        cells[1] = new Cell(row, col + 1);
        cells[2] = new Cell(row, col + 2);
        cells[3] = new Cell(row + 1, col + 1);
    }

    /**
     * 按顺时针方向，打印方块中四个格子所在的坐标
     */
    public void print() {
        String str = "";
        for (int i = 0; i < cells.length - 1; i++) {
            str += "(" + cells[i].getCellInfo() + ", ";
        }
        str += "(" + cells[cells.length - 1].getCellInfo() + ")";
        System.out.println(str);
    }
}

```

```

    * 使方块下落一个格子
    */
    public void drop() {
        for (int i = 0; i < cells.length; i++) {
            cells[i].row++;
        }
    }

    /**
     * 使方块左移一个格子
     */
    public void moveLeft() {
        for (int i = 0; i < cells.length; i++) {
            cells[i].col--;
        }
    }

    /**
     * 使方块右移一个格子
     */
    public void moveRight() {
        for (int i = 0; i < cells.length; i++) {
            cells[i].col++;
        }
    }
}

```

TestT 类完整代码如下所示：

```

public class TestT {
    public static void main(String[] args) {
        T t=new T(0,1);
        //测试 print 方法
        System.out.println("原始坐标为: ");
        t.print();

        //测试 drop 方法
        //    t.drop();
        //    System.out.println("调用 drop 方法后的坐标: ");
        //    t.print();

        //测试 moveLeft 方法
        //    t.moveLeft();
        //    System.out.println("调用 moveLeft 方法后的坐标: ");
        //    t.print();

        //测试 moveRight 方法
        t.moveRight();
        System.out.println("调用 moveRight 方法后的坐标: ");
        t.print();
    }
}

```

J 类的完整代码如下所示：

```

public class J {
    Cell[] cells;// 属性,用来存储一个方块的四个格子的坐标

    /**
     * 构造方法, 为属性 cells 进行初始化
     */
    public J() {

```

```

        this(0, 0);
    }

    /**
     * 构造方法，为属性 cells 进行初始化
     *
     * @param row
     *          顺时针方向，第一个坐标的行
     * @param col
     *          顺时针方向，第一个坐标的列
     */
    public J(int row, int col) {
        cells = new Cell[4];
        // 按顺时针方向初始化 Cell
        cells[0] = new Cell(row, col);
        cells[1] = new Cell(row, col + 1);
        cells[2] = new Cell(row, col + 2);
        cells[3] = new Cell(row + 1, col + 2);
    }

    /**
     * 按顺时针方向，打印方块中四个格子所在的坐标
     */
    public void print() {
        String str = "";
        for (int i = 0; i < cells.length - 1; i++) {
            str += "(" + cells[i].getCellInfo() + "), ";
        }
        str += "(" + cells[cells.length - 1].getCellInfo() + ")";
        System.out.println(str);
    }

    /**
     * 使方块下落一个格子
     */
    public void drop() {
        for (int i = 0; i < cells.length; i++) {
            cells[i].row++;
        }
    }

    /**
     * 使方块左移一个格子
     */
    public void moveLeft() {
        for (int i = 0; i < cells.length; i++) {
            cells[i].col--;
        }
    }

    /**
     * 使方块右移一个格子
     */
    public void moveRight() {
        for (int i = 0; i < cells.length; i++) {
            cells[i].col++;
        }
    }
}

```

TestJ 类的完整代码如下所示：

```

public class TestJ {
    public static void main(String[] args) {
        J j=new J(17,3);
    }
}

```

```
//测试print方法
System.out.println("原始坐标为: ");
j.print();

//测试drop方法
//    j.drop();
//    System.out.println("调用 drop 方法后的坐标: ");
//    j.print();

//测试moveLeft方法
//    j.moveLeft();
//    System.out.println("调用 moveLeft 方法后的坐标: ");
//    j.print();

//测试moveRight方法
j.moveRight();
System.out.println("调用 moveRight 方法后的坐标: ");
j.print();
}
```

达内IT培训

课后作业

1. 请描述下列代码的运行结果

```
public class ExerciseTest {  
    public static void main(String[] args){  
        ExerciseTest f = new ExerciseTest();  
        System.out.println(f.add("4", "5"));  
    }  
  
    public int add(int x, int y) {  
        return x + y;  
    }  
    public String add(String x, String y) {  
        return x + y;  
    }  
}
```

2. 关于构造方法，下面说法正确的是

- A. 构造方法不能带有参数
- B. 构造方法的名称必须和类名相同
- C. 构造方法可以定义返回值
- D. 构造方法不能重载

3. 定义 Tetris 项目中的 O 类并测试

在今天的课上案例中已经定义了 T 型和 J 型方块对应的类，本案例要求模仿课上定义的 T 型方块的类 T 和 J 型方块的类 J，来定义 O 型方块对应的类 O。O 型方块的形状如图-1 中的红色方块。

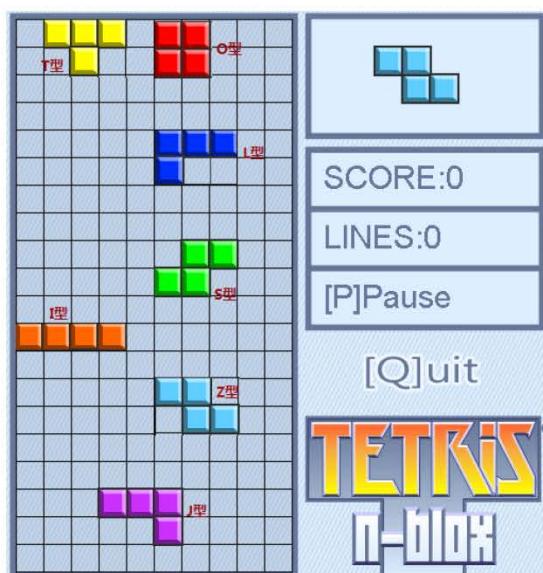


图- 1

Java 面向对象

Unit03

知识体系.....Page 62

对象内存管理	对象内存管理	对象内存管理
堆内存		对象存储在堆中
		成员变量的生命周期
		垃圾回收机制
		Java 程序的内存泄露问题
		System.gc()方法
非堆—栈		栈用于存放方法中的局部变量
		局部变量的生命周期
		成员变量和局部变量
非堆—方法区		方法区用于存放类的信息
		方法只有一份
继承	继承	泛化的过程
		extends 关键字
		继承中构造方法
		父类的引用指向子类的对象

经典案例.....Page 69

构建 Tetromino 类，重构 T 和 J 类并测试	泛化的过程
	extends 关键字
	继承中构造方法
	父类的引用指向子类的对象

课后作业.....Page 77

1. 对象内存管理

1.1. 对象内存管理

1.1.1. 【对象内存管理】对象内存管理

Tarena
达内科技

对象内存管理

- 编译好的Java程序需要运行在JVM中。
- 程序，无论代码还是数据，都需要存储在内存中。JVM为Java程序提供并管理所需要的内存空间。
- JVM内存分为“堆”、“栈”和“方法区”三个区域，分别用于存储不同的数据。

知识讲解

The diagram shows the JVM's memory structure with three main components: '栈' (Stack) represented by a tall narrow rectangle, '堆' (Heap) represented by a wide rectangle below it, and '方法区' (Method Area) represented by a rectangle above the heap. The stack and method area are shown as light gray rectangles, while the heap is a darker gray.

1.2. 堆内存

1.2.1 【堆内存】对象存储在堆中

The diagram illustrates the creation of a new object in Java memory. On the left, a vertical blue bar labeled "知识讲解" (Knowledge Lecture) contains a plus sign (+). To its right, a horizontal line represents the stack memory, with a variable `c` pointing to the memory address `40DF`. To the right of the stack is a larger rounded rectangle representing the heap memory. Inside the heap, a `Cell` object is shown with two fields: `row` and `col`, both initialized to 0.

对象存储在堆中

- JVM在其内存空间开辟一个称为“堆”的存储空间；
- 这部分空间用于存储使用new关键字所创建的对象。

```
Cell c = new Cell();
```

1.2.2 【堆内存】成员变量的生命周期

成员变量的生命周期

- 访问对象需要依靠引用变量。
- 当一个对象没有任何引用时，被视为废弃的对象，属于被回收的范围。该对象中的所有成员变量也随之被回收。
- 成员变量的生命周期为：从对象在堆中创建开始到对象从堆中被回收结束。

知识讲解

```
Cell c = new Cell();  
c = null;  
//不再指向刚分配的对象空间，成员变量失效
```

+

1.2.3. 【堆内存】垃圾回收机制

垃圾回收机制

- 垃圾回收器 (Garbage Collection , GC) 是JVM自带的一个线程 (自动运行着的程序) , 用于回收没有任何引用指向的对象。
- Java程序员不用担心内存管理 , 因为垃圾收集器会自动进行回收管理。

知识讲解

++

1.2.4. 【堆内存】Java 程序的内存泄露问题

Technologies
Tarena
达内科技

Java程序的内存泄漏问题

- 内存泄漏是指，不再使用的内存没有被及时的回收。严重的内存泄漏会因过多的内存占用而导致程序的崩溃。
- GC线程判断对象是否可以回收的依据是该对象是否有引用指向，因此，当确定该对象不再使用时，应该及时将其引用设置为null。

知识讲解

1.2.5. 【堆内存】System.gc()方法

Technology
Tarena
达内科技

System.gc()方法

- GC的回收对程序员来说是透明的，并不一定一发现有无引用的对象，就立刻回收。
- 一般情况下，当我们需要GC线程即刻回收无用对象时，可以调用 System.gc()方法。
- System.gc() 用于建议虚拟机马上调度GC线程回收资源，具体的实现策略取决于不同的JVM系统。

知识讲解

1.3. 非堆—栈

1.3.1. 【非堆—栈】栈用于存放方法中的局部变量

知识讲解

- JVM在其内存空间开辟一个称为“栈”的存储空间；
- 这部分空间用于存储程序运行时在方法中声明的所有局部变量。例如：main() 方法中有如下代码：

```
Cell c = new Cell();  
int num = 5;
```

The diagram shows the memory layout. On the left, a vertical rectangle represents the stack (栈内存), containing two slots: one for 'num' with value 5 and address 40DF, and another for 'c'. An arrow points from 'c' to the right, where a larger rounded rectangle represents the heap (堆内存). Inside the heap, there is an object of type 'Cell'. This object has two fields: 'row' and 'col', both of which are 0.

1.3.2. 【非堆—栈】局部变量的生命周期

Tarena
达内科技

局部变量的生命周期

- 一个运行的Java程序从开始到结束会有多次方法的调用。JVM会为每一个方法的调用在栈中分配一个对应的空间，这个空间称为该方法的栈帧。
- 一个栈帧对应一个正在调用中的方法，栈帧中存储了该方法的参数、局部变量等数据。当某一个方法调用完成后，其对应的栈帧将被清除，局部变量失效。

知识讲解

1.3.3. 【非堆—栈】成员变量和局部变量

Technology
Tarena
达内科技

成员变量和局部变量

- 成员变量和局部变量的差别如下：
- 局部变量：
 - 定义在方法中；
 - 没有默认值，必须自行设定初始值；
 - 方法被调用时，存在栈中，方法调用结束，从栈中清除；
- 成员变量：
 - 定义在类中，方法外；
 - 有默认初始值，可以不显式初始化；
 - 所在类被实例化后，存在堆中，对象被回收时，成员变量失效；

知识讲解

1.4. 非堆—方法区

1.4.1. 【非堆—方法区】方法区用于存放类的信息

方法区用于存放类的信息

- 方法区用于存放类的信息，Java程序运行时，首先会通过类装载器载入类文件的字节码信息，经过解析后将其装入方法区。类的各种信息（包括方法）都在方法区存储。

Cell c = new Cell()

Cell类首先被装载到JVM的方法区，其中包括类的基本信息和方法定义等。

Technology **Tarena**
达内科技

方法区

Cell.class.drop()

栈

堆

c

Cell

row = 0
col = 0

1.4.2. 【非堆—方法区】方法只有一份

Technology
Tarena
达内科技

方法只有一份

- 当类的信息被加载到方法区时，除了类的类型信息以外，同时类内的方法定义也被加载到方法区；
- 类在实例化对象时，多个对象会拥有各自在堆中的空间，但所有实例对象是共用在方法区中的一份方法定义的。

知识讲解

```
JFrame f1 = new JFrame( );
```

```
JFrame f2 = new JFrame( );
```

```
f1.setSize(200, 300);
```

```
f2.setSize(300,400);
```

setSize方法只有一份，分别针对f1指向的对象和f2指向的对象调用了两次。

2. 继承

2.1. 继承

2.1.1. 【继承】泛化的过程

The diagram illustrates the process of generalization (泛化) in object-oriented programming. It shows two specific classes, "T型类" (T Type Class) and "J型类" (J Type Class), which inherit from a general class, "Tetromino".

泛化的过程

知识讲解

T型类
cells[4]
drop()
moveLeft()
moveRight()
print()

J型类
cells[4]
drop()
moveLeft()
moveRight()
print()

Tetromino (父类)
cells[4]
drop()
moveLeft()
moveRight()
print()

四格拼板 (Speech bubble)

子类

A large arrow points from the "Tetromino" box down to the "T型类" and "J型类" boxes, indicating inheritance.

2.1.2. 【继承】extends 关键字

Tarena
达内科技

extends关键字

- 通过extends关键字可以实现类的继承；
- 子类 (Sub class) 可以继承父类 (Super class) 的成员变量及成员方法，同时也可定义自己的成员变量和成员方法；
- Java语言不支持多重继承，一个类只能继承一个父类，但一个父类可以有多个子类。

知识讲解

Tarena
达内科技

extends关键字 (续1)

```
public class Tetromino {
```

```
    Cell[ ] cells;
```

```
    public Tetromino() {
```

```
        cells = new Cell[4];
```

```
}
```

```
    public void drop() {...}
```

```
    public void moveLeft() {...}
```

```
    public void moveRight() {...}
```

```
    public void print() {...}
```

```
}
```

Tarena
达内科技

extends关键字 (续2)

```
public class TetrominoT extends Tetromino {
```

```
    public TetrominoT(int row, int col) {
```

```
        cells[0] = new Cell(row, col);
```

```
        cells[1] = new Cell(row, col + 1);
```

```
        cells[2] = new Cell(row, col + 2);
```

```
        cells[3] = new Cell(row + 1, col + 1);
```

```
}
```

```
}
```

通过继承，TetrominoT拥有Cell[] cells成员以及drop、moveLeft、moveRight等方法。

2.1.3. 【继承】继承中构造方法

继承中构造方法

• 子类的构造方法中必须通过super关键字调用父类的构造方法，这样可以妥善的初始化继承自父类的成员变量。

• 如果子类的构造方法中没有调用父类的构造方法，Java编译器会自动的加入对父类无参构造方法的调用（如果该父类没有无参的构造方法，会有编译错误）。

```
public TetrominoT(int row, int col) {
    super(); //编译器自动加入
    cells[0] = new Cell(row, col);
    cells[1] = new Cell(row, col + 1);
    .....
} //super关键字必须位于子类构造方法的第一行
```

+

继承中构造方法（续1）

```
class Foo {
    int value;
    Foo(int value) {
        this.value = value;
    }
}
```

```
class Goo extends Foo {
    int num;
    Goo(int num) {
        this.num = num;
    }
}
```

编译错误，子类的构造方法中没有调用父类的构造方法（虽然有默认super()，但是父类中没有定义无参的构造方法）。

编译正确，子类中调用了父类的构造方法，初始化了继承自父类的value成员变量。

+

2.1.4. 【继承】父类的引用指向子类的对象

父类的引用指向子类的对象

• 一个子类的对象可以向上造型为父类的类型。即，定义父类型的引用可以指向子类的对象。

```
class Foo {
    int value;
    public void f() {...}
    Foo(int value) {
        this.value = value;
    }
}
```

```
class Goo extends Foo {
    int num;
    public void g() {...}
    Goo(int value, int num) {
        super(value);
        this.num = num;
    }
}
```

Foo obj = new Goo(100, 3);

+

父类的引用指向子类的对象（续1）

- 父类的引用可以指向子类的对象，但通过父类的引用只能访问父类所定义的成员，不能访问子类扩展的部分。

```
class Foo {  
    int value;  
    public void f() {...}  
    ... ... ...  
}
```

```
class Goo extends Foo {  
    int num;  
    public void g() {...}  
    ... ... ...  
}
```

```
Foo obj = new Goo(100,3);  
obj.value=200;  
obj.f();  
obj.num = 5;  
obj.g();
```

会有编译错误，Java编译器会根据引用的类型（ Foo ），而不是对象的类型（ Goo ）来检查调用的方法是否匹配。



经典案例

1. 构建 Tetromino 类，重构 T 和 J 类并测试

- 问题

分析案例“定义 Tetris 项目中的 T 类和 J 类并测试”中的 T 类和 J 类，会发现存在大量的重复代码，比如，cells 属性、print 方法、drop 方法、moveLeft 方法、moveRight 方法，这四个方法在各个类中的实现都是相同的。因此，本案例要求使用继承的方式，构建 T 类和 J 类父类 Tetromino 类，重构 T 类和 J 类并测试重构后的代码。

另外，测试时，需要打印出游戏所在的平面（宽 10 格，高 20 格），用“-”号表示平面上的每个单元；然后使用“*”号打印显示方块中的每个格子，如图-1 中所示。

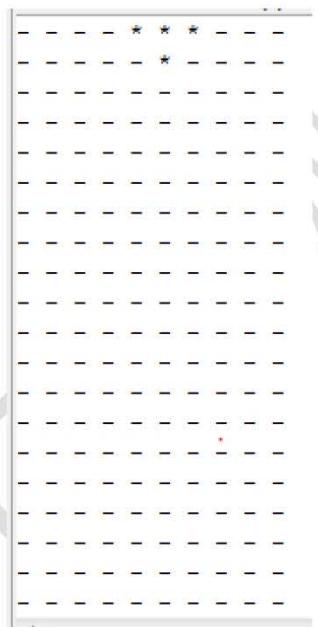


图- 1

- 方案

要实现本案例的功能，解决方案如下：

1. 抽取出，T 类和 J 类中共有的属性和方法，然后，构建 Tetromino 类，将这些共有的属性和方法抽取到 Tetromino 类中。
2. 重构 T 类为 TetrominoT 类。在 TetrominoT 类中，只保留该类特有的部分，比如，该类的构造中，初始化 T 型方块部分。
3. 重构 J 类为 TetrominoJ 类。在 TetrominoJ 类中，也是只保留该类特有的部分，比如，该类的构造中，初始化 J 型方块部分。
4. 测试重构后的代码，构建 TetrominoGame 类。首先，在 TetrominoGame 类中，创建方法打印出游戏所在的平面（宽 10 格，高 20 格），用“-”号表示平面上的每个

单元；然后使用“*”号打印显示方块中的每个格子。

• 步骤

实现此案例需要按照如下步骤进行。

步骤一：构建 Tetromino 类

首先，抽取出 T 类和 J 类中共有的属性和方法，然后，构建 Tetromino 类，将这些共有的属性和方法抽取到 Tetromino 类中。抽取 T 类和 J 类中的 cells 属性，print 方法、drop 方法、moveLeft 方法以及 moveRight 方法到 Tetromino 类中，代码如下所示：

```
public class Tetromino {  
    Cell[] cells; // 属性，用来存储一个方块的四个格子的坐标  
  
    /**  
     * 按顺时针方向，打印方块中四个格子所在的坐标  
     */  
    public void print() {  
        String str = "";  
        for (int i = 0; i < cells.length - 1; i++) {  
            str += "(" + cells[i].getCellInfo() + "), ";  
        }  
        str += "(" + cells[cells.length - 1].getCellInfo() + ")";  
        System.out.println(str);  
    }  
  
    /**  
     * 使方块下落一个格子  
     */  
    public void drop() {  
        for (int i = 0; i < cells.length; i++) {  
            cells[i].row++;  
        }  
    }  
  
    /**  
     * 使方块左移一个格子  
     */  
    public void moveLeft() {  
        for (int i = 0; i < cells.length; i++) {  
            cells[i].col--;  
        }  
    }  
  
    /**  
     * 使用方块右移一个格子  
     */  
    public void moveRight() {  
        for (int i = 0; i < cells.length; i++) {  
            cells[i].col++;  
        }  
    }  
}
```

步骤二：定义 Tetromino 类的构造方法

查看 T 类和 J 类带参构造方法中，都对 cells 数组进行了初始化，因此，对 cells 数

组的初始化也属于两个类的共有部分。所以，定义 Tetromino 类的无参数构造方法，在构造方法中，初始化 cells 数组的长度为 4，代码如下所示：

```
public class Tetromino {  
    Cell[] cells; // 属性，用来存储一个方块的四个格子的坐标  
  
    /**  
     * 构造方法，初始化 cells 数组  
     */  
    public Tetromino() {  
        cells = new Cell[4];  
    }  
  
    /**  
     * 按顺时针方向，打印方块中四个格子所在的坐标  
     */  
    public void print() {  
        String str = "";  
        for (int i = 0; i < cells.length - 1; i++) {  
            str += "(" + cells[i].getCellInfo() + "), ";  
        }  
        str += "(" + cells[cells.length - 1].getCellInfo() + ")";  
        System.out.println(str);  
    }  
  
    /**  
     * 使方块下落一个格子  
     */  
    public void drop() {  
        for (int i = 0; i < cells.length; i++) {  
            cells[i].row++;  
        }  
    }  
  
    /**  
     * 使方块左移一个格子  
     */  
    public void moveLeft() {  
        for (int i = 0; i < cells.length; i++) {  
            cells[i].col--;  
        }  
    }  
  
    /**  
     * 使用方块右移一个格子  
     */  
    public void moveRight() {  
        for (int i = 0; i < cells.length; i++) {  
            cells[i].col++;  
        }  
    }  
}
```

步骤三：重构 T 类

重构 T 类为 TetrominoT 类，在步骤一和步骤二中，我们已经将 T 类和 J 类的共有部分抽取出去。再此，重构 T 类时，只有构造方法的实现不同。构造方法的实现为根据不同的

形状给 cells 数组的每个元素进行赋值。TetrominoT 类的代码如下所示：

```
public class TetrominoT extends Tetromino {  
    public TetrominoT(int row, int col) {  
        super();  
        // 按顺时针方向初始化 Cell  
        cells[0] = new Cell(row, col);  
        cells[1] = new Cell(row, col + 1);  
        cells[2] = new Cell(row, col + 2);  
        cells[3] = new Cell(row + 1, col + 1);  
    }  
}
```

上述代码中，使用 super 关键字调用父类的无参数构造方法。代码 “super();” 是可以省略不写的。默认情况下，系统在子类构造方法的第一句代码就是 “super();” 即，调用父类无参数的构造方法。

步骤四：重构 J 类

重构 J 类为 TetrominoJ 类，重构 J 类时，和重构 T 是一样的实现，只有构造方法的实现不同。TetrominoJ 类的代码如下所示：

```
public class TetrominoJ extends Tetromino {  
    public TetrominoJ(int row, int col) {  
        cells[0] = new Cell(row, col);  
        cells[1] = new Cell(row, col + 1);  
        cells[2] = new Cell(row, col + 2);  
        cells[3] = new Cell(row + 1, col + 2);  
    }  
}
```

上述代码中，在 TetrominoJ 类的构造方法的第一句，虽然没有使用 super 关键字调用父类无参数的构造方法，但是，系统会默认调用父类无参数的构造方法。

步骤五：测试重构后的代码

测试重构后的代码，构建 TetrominoGame 类。首先，在 TetrominoGame 类中，创建方法打印出游戏所在的平面（宽 10 格，高 20 格），用 “-” 号表示平面上的每个单元格；然后使用 “*” 号打印显示方块中的每个格子。TetrominoGame 类的代码如下所示：

```
import java.util.Scanner;  
  
public class TetrominoGame {  
    /**  
     * 打印出游戏所在的平面（宽 10 格，高 20 格）。用“-”号表示平面上的每个单元，用“*”号打印  
     * 显示方块中的每个格子  
     *  
     * @param tetromino 需要显示在游戏平面中的方块  
     */  
    public static void printTetromino(Tetromino tetromino) {  
        int totalRow = 20;  
        int totalCol = 10;  
        // 获取方块中存储的四个格子的数组  
        Cell[] cells = tetromino.cells;  
        for (int row = 0; row < totalRow; row++) {  
            for (int col = 0; col < totalCol; col++) {  
                if (cells[row % 4][col % 4] == true) {  
                    System.out.print("*");  
                } else {  
                    System.out.print("-");  
                }  
            }  
            System.out.println();  
        }  
    }  
}
```

```

        for (int col = 0; col < totalCol; col++) {
            // 用于判断该位置是否包含在 cells 数组中
            boolean isInCells = false;
            for (int i = 0; i < cells.length; i++) {
                if (cells[i].row == row && cells[i].col == col) {
                    System.out.print("* ");
                    isInCells = true;
                    break;
                }
            }
            if (!isInCells) {
                System.out.print("- ");
            }
        }
        System.out.println();
    }
}

```

在 TetrominoGame 类中，添加 main 方法，在控制台，打印 T 型和 J 型。代码如下所示：

```

public class TetrominoGame {

    public static void main(String[] args) {
        // 测试 TetrominoT
        System.out.println("-----打印 T 型-----");
        Tetromino t = new TetrominoT(0, 4);
        printTetromino(t);

        // 测试 TetrominoJ
        System.out.println("-----打印 J 型-----");
        Tetromino j = new TetrominoJ(0, 4);
        printTetromino(j);
    }

    /**
     * 打印出游戏所在的平面（宽 10 格，高 20 格）。用“-”号表示平面上的每个单元，用“*”号打印显示方块中的每个格子
     */
    * @param tetromino 需要显示在游戏平面中的方块
    */
    public static void printTetromino(Tetromino tetromino) {
        int totalRow = 20;
        int totalCol = 10;
        // 获取方块中存储的四个格子的数组
        Cell[] cells = tetromino.cells;
        for (int row = 0; row < totalRow; row++) {
            for (int col = 0; col < totalCol; col++) {
                // 用于判断该位置是否包含在 cells 数组中
                boolean isInCells = false;
                for (int i = 0; i < cells.length; i++) {
                    if (cells[i].row == row && cells[i].col == col) {
                        System.out.print("* ");
                        isInCells = true;
                        break;
                    }
                }
                if (!isInCells) {
                    System.out.print("- ");
                }
            }
            System.out.println();
        }
    }
}

```

```
        }
        if (!isInCells) {
            System.out.print("- ");
        }
    }
    System.out.println();
}
}
```

控制台输出结果如下所示：

从上述代码和打印结果可以看出，父类的引用是可以指向子类的对象的。构造子类对象时，也构造了父类的对象。

• 完整代码

本案例中，Tetromino 类的完整代码如下所示：

```

public class Tetromino {
    Cell[] cells; // 属性, 用来存储一个方块的四个格子的坐标

    /**
     * 构造方法, 初始化 cells 数组
     */
    public Tetromino() {
        cells = new Cell[4];
    }
    /**
     * 按顺时针方向, 打印方块中四个格子所在的坐标
     */
    public void print() {
        String str = "";
        for (int i = 0; i < cells.length - 1; i++) {
            str += "(" + cells[i].getCellInfo() + "), ";
        }
        str += "(" + cells[cells.length - 1].getCellInfo() + ")";
        System.out.println(str);
    }

    /**
     * 使方块下落一个格子
     */
    public void drop() {
        for (int i = 0; i < cells.length; i++) {
            cells[i].row++;
        }
    }

    /**
     * 使方块左移一个格子
     */
    public void moveLeft() {
        for (int i = 0; i < cells.length; i++) {
            cells[i].col--;
        }
    }

    /**
     * 使用方块右移一个格子
     */
    public void moveRight() {
        for (int i = 0; i < cells.length; i++) {
            cells[i].col++;
        }
    }
}

```

TetrominoT 类的完整代码如下所示：

```

public class TetrominoT extends Tetromino {
    public TetrominoT(int row, int col) {
        super();
        // 按顺时针方向初始化 Cell
        cells[0] = new Cell(row, col);
        cells[1] = new Cell(row, col + 1);
        cells[2] = new Cell(row, col + 2);
        cells[3] = new Cell(row + 1, col + 1);
    }
}

```

TetrominoJ 类的完整代码如下所示：

```
public class TetrominoJ extends Tetromino {  
    public TetrominoJ(int row, int col) {  
        cells[0] = new Cell(row, col);  
        cells[1] = new Cell(row, col + 1);  
        cells[2] = new Cell(row, col + 2);  
        cells[3] = new Cell(row + 1, col + 2);  
    }  
}
```

TetrominoGame 类的完整代码如下所示：

```
public class TetrominoGame {  
    public static void main(String[] args) {  
        //测试 TetrominoT  
        System.out.println("-----打印 T 型-----");  
        Tetromino t = new TetrominoT(0, 4);  
        printTetromino(t);  
  
        //测试 TetrominoJ  
        System.out.println("-----打印 J 型-----");  
        Tetromino j = new TetrominoJ(0, 4);  
        printTetromino(j);  
    }  
  
    /**  
     * 打印出游戏所在的平面（宽 10 格，高 20 格）。用“—”号表示平面上的每个单元，用“*”号打印  
     * 显示方块中的每个格子  
     */  
    * @param tetromino 需要显示在游戏平面中的方块  
    */  
    public static void printTetromino(Tetromino tetromino) {  
        int totalRow = 20;  
        int totalCol = 10;  
        //获取方块中存储的四个各自的数组  
        Cell[] cells = tetromino.cells;  
        for (int row = 0; row < totalRow; row++) {  
            for (int col = 0; col < totalCol; col++) {  
                // 用于判断该位置是否包含在 cells 数组中  
                boolean isInCells = false;  
                for (int i = 0; i < cells.length; i++) {  
                    if (cells[i].row == row && cells[i].col == col) {  
                        System.out.print("* ");  
                        isInCells = true;  
                        break;  
                    }  
                }  
                if (!isInCells) {  
                    System.out.print("- ");  
                }  
            }  
            System.out.println();  
        }  
    }  
}
```

课后作业

1. 简述 JVM 垃圾回收机制
2. Java 程序是否会出现内存泄露
3. JVM 如何管理内存，分成几个部分？分别有什么用途？说出下面代码的内存实现原理：

```
Foo foo = new Foo();
foo.f();
```

4. 指出下面代码的编译错误，并说明原因

```
//哺乳动物
public class Mammals {
}

//鸟类
public class Birds {

}

//蝙蝠
public class Bat extends Mammals,Birds{}
```

5. 说出下面代码的输出结果，并解释原因

```
public class Sub extends Base {
    String color;

    public Sub(double size, String name, String color) {
        super(size, name);
        this.color = color;
    }

    public static void main(String[] args) {
        Sub s = new Sub(5.6, "测试对象", "红色");
        System.out.println(s.size + "----" + s.name + "----" + s.color);
    }
}

class Base {
    double size;
    String name;

    public Base(double size, String name) {
        this.size = size;
        this.name = name;
    }
}
```

Java 面向对象

Unit04

知识体系.....Page 79

继承	重写	方法的重写
		重写中使用 super 关键字
		重写和重载的区别
访问控制	包的概念	package 语句
		import 语句
	访问控制修饰符	封装的意义
		public 和 private
		protected 和默认访问控制
		访问控制符修饰类
		访问控制符修饰成员
static 和 final	static 关键字	static 修饰成员变量
		static 修饰方法
		static 块
	final 关键字	final 修饰变量
		final 修饰方法
		final 修饰类
		static final 常量

经典案例.....Page 87

重写 T 类和 J 类的 print 方法并测试	方法的重写
	重写中使用 super 关键字

课后作业.....Page 94

1. 继承

1.1. 重写

1.1.1. 【重写】方法的重写

方法的重写

- 子类可以重写(覆盖)继承自父类的方法,即方法名和参数列表与父类的方法相同;但方法的实现不同。
- 当子类对象的重写方法被调用时(无论是通过子类的引用调用还是通过父类的引用调用),运行的是子类的重写后的版本。

方法的重写 (续1)

```
class Foo {  
    public void f() {  
        System.out.println("Foo.f()");  
    }  
}  
  
class Goo extends Foo {  
    public void f() {  
        System.out.println("Goo.f()");  
    }  
}
```

知识讲解

Goo obj1 = new Goo();
obj1.f();
Goo obj2 = new Goo();
obj2.f();

输出的结果均为“Goo.f()”，当子类对象的重写方法被调用时（无论是通过子类的引用调用还是通过父类的引用调用），运行的是子类的重写后的版本。

1.1.2. 【重写】重写中使用 super 关键字

Technology
Tarena
达内科技

重写中使用super关键字

- 子类在重写父类的方法时，可以通过super关键字调用父类的版本。

```
class Foo {  
    public void f0() {  
        System.out.println("Foo.f0()");  
    }  
}
```

知识讲解

```
class Goo extends Foo {  
    public void f0() {  
        super.f0();  
        System.out.println("Goo.f0()");  
    }  
}
```

```
Foo obj2 = new Goo();  
obj2.f0();
```

输出的结果为：
Foo.f0();
Goo.f0();
子类重写方法中的super.f0();
调用了父类的版本；这样的语法通常用于子类的重写方法在父类方法的基础之上进行功能扩展。

1.1.3. 【重写】重写和重载的区别

Innovate
Tarena
达内科技

重写和重载的区别

- 重载与重写是完全不同的语法现象，区别如下：
 - 重载是指在一个类中定义多个方法名相同但参数列表不同的方法，在编译时，根据参数的个数和类型来决定绑定哪个方法。
 - 重写是指在子类中定义和父类完全相同的方法，在程序运行时，根据对象的类型不同（而不是引用类型）而调用不同的版本。

知识讲解

Tarena
达内科技

重写和重载的区别 (续1)

```
class Super {  
    public void f() {  
        System.out.println("super.f()");  
    }  
}  
class Sub extends Super {  
    public void f() {  
        System.out.println("sub.f()");  
    }  
}  
class Goo {  
    public void g(Super obj) {  
        System.out.println("g(Super)");  
        obj.f();  
    }  
    public void g(Sub obj) {  
        System.out.println("g(Sub)");  
        obj.f();  
    }  
}
```

知识讲解

```
Super obj = new Sub();  
Goo goo = new Goo();  
goo.g(obj);
```

输出结果是：
g(Super)
sub.f()

Tarena
达内科技

重写和重载的区别 (续2)

- 重载遵循所谓“编译期绑定”，即在编译时根据参数变量的类型判断应该调用哪个方法；因为：变量obj的类型为Super，因此：Goo的g(Super)方法被调用。
- 重写遵循所谓“运行期绑定”，即在运行的时候根据引用变量指向的实际对象类型调用方法；因为：obj实际指的是子类的对象，因此：子类重写后的f方法被调用。

知识讲解

2. 访问控制

2.1. 包的概念

2.1.1. 【包的概念】package 语句

Technology
Tarena
达内科技

package语句

- 定义类时需要指定类的名称，但如果仅仅将类名作为类的唯一标识，则不可避免的出现命名冲突的问题，这会给组件复用以及团队间的合作造成很大的麻烦！
- 在Java语言中，用包（ package ）的概念来解决命名冲突的问题。在定义一个类时，除了定义类的名称一般还要指定一个包名，定义包名的语法为： package 包名；

知识讲解

例如：

```
package test;  
class Point {  
    ... ... ...  
}
```

package语句必须写在Java源文件的最开始，在类定义之前。例如：下面的语句将为Point类指定包名为“test”：一旦使用的package指定了包名，类的全称应该是“包名.类名”。例如，上述的Point类的全称是test.Point。不同的包中可以定义相同的类名，例如test1.Point和test2.Point是两个截然不同的名称。

Tarena
达内科技

package语句 (续1)

- 包名也可以有层次结构，在一个包中可以包含另外一个包。可以按照如下方式写package语句：

```
package 包名1.包名2...包名n;
```

- 如果各个公司或开发组织的程序员都随心所欲的命名包名的话，仍然不能从根本上解决命名冲突的问题。因此，在指定包名的时候应该按照一定的规范，例如：

```
org.apache.commons.lang.StringUtils
```

StringUtils是类名。而org.apache.commons.lang是多层包名，其含义如下：org.apache表示公司或组织的信息（是这个公司（或组织）域名的反写）；commons表示项目的名称信息；lang表示模块的名称信息。

知识讲解

+

2.1.2. 【包的概念】import 语句

University
Tarena
达内科技

import语句

- 访问一个类时需要使用该类的全称，但这样的书写方式过于繁琐；
- 可以通过import语句对类的全称进行声明。
import语句的语法为：
import 类的全局限定名
(即包名+类名)；
- 通过import语句声明了类的全称后，该源文件中就可以直接使用类名来访问了。

知识讲解

```
org.whatisjava.core.Point p =  
    new org.whatisjava.core.Point();
```

```
import org.whatisjava.core.Point;
```

```
import org.whatisjava.core.Point;  
public class Main {  
    public static void main(String[] args) {  
        Point p = new Point(100, 200);  
    }  
}
```

2.2. 访问控制修饰符

2.2.1. 【访问控制修饰符】封装的意义

知识讲解



采用哪一种？ 前提：来的都是活雷锋！

+
+
+
+

Tarena
达内科技

封装的意义（续1）

- 对外提供可调用的、稳定的功能；
- 封装容易变化的、具体的实现细节，外界不可访问，这样的意义在于：
 - 降低代码出错的可能性，便于维护；
 - 当内部的实现细节改变时，只要保证对外的功能定义不变，其他的模块就不会因此而受到牵连；

知识讲解

2.2.2. 【访问控制修饰符】public 和 private

Technology
Tarena
达内科技

public和private

- private修饰的成员变量和方法仅仅只能在本类中调用；
public修饰的成员变量和方法可以在任何地方调用。
public修饰的内容是对外提供可以被调用的功能，需要相对稳定；private修饰的内容是对内实现的封装，如果“公开”会增加维护的成本。

知识讲解

```
public class Point {  
    private int x;  
    private int y;  
    Point(int x, int y) {...}  
    public int distance(Point p) {...}  
}
```

```
Point p1 = new Point(1, 2);  
Point p2 = new Point(3, 4);  
p1.x = 100
```

```
\ The field Point.x is not visible  
int d = p1.distance(p2);
```

2.2.3. 【访问控制修饰符】protected 和默认访问控制

protected和默认访问控制

Tarena
达内科技

- 用protected修饰的成员变量和方法可以被子类及同一个包中的类使用。
- 默认访问控制即不书写任何访问控制符。默认访问控制的成员变量和方法可以被同一个包中的类调用

+

2.2.4. 【访问控制修饰符】访问控制符修饰类

访问控制符修饰类

Tarena
达内科技

- 对于类的修饰可以使用public和默认方式。public修饰的类可以被任何一个类使用；默认访问控制的类只可以被同一个包中的类使用。
- protected和private可以用于修饰内部类。

+

2.2.5. 【访问控制修饰符】访问控制符修饰成员

访问控制符修饰成员

Tarena
达内科技

- 访问控制符修饰成员时的访问权限如下表所示：

修饰符	本类	同一个包中的类	子类	其他类
public	可以访问	可以访问	可以访问	可以访问
protected	可以访问	可以访问	可以访问	不能访问
默认	可以访问	可以访问	不能访问	不能访问
private	可以访问	不能访问	不能访问	不能访问

+

3. static 和 final

3.1. static 关键字

3.1.1. 【static 关键字】 static 修饰成员变量

Technology
Tarena
达内科技

static修饰成员变量

- 用static修饰的成员变量不属于对象的数据结构；
- static变量是属于类的变量，通常可以通过类名来引用static成员；
- static成员变量和类的信息一起存储在方法区，而不是在堆中，一个类的static成员变量只有“一份”，无论该类创建了多少对象。

知识讲解

Tarena
达内科技

static修饰成员变量 (续1)

```
class Cat {  
    private int age;  
    private static int numOfCats;  
    public Cat(int age) {  
        this.age = age;  
        System.out.println(  
            ++numOfCats);  
    }  
}
```

```
Cat c1 = new Cat( 2);  
Cat c2 = new Cat( 3);
```

知识讲解

The diagram illustrates the state of memory after creating two `Cat` objects. It shows the stack and heap memory areas.

- Stack Memory:** Contains two pointers, `c2` and `c1`, which point to the objects in the heap.
- Heap Memory:** Contains two objects, each with an `age` field.
 - The first object (pointed by `c2`) has an `age` of 3.
 - The second object (pointed by `c1`) has an `age` of 2.
- Variable numofCats:** Located in the stack, its value is 2.

3.1.2. 【static 关键字】 static 修饰方法

Tarena
达内科技

static修饰方法

- 通常的方法都会涉及到对具体对象的操作，这些方法在调用时，需要隐式的传递对象的引用（this）。

```
int d = p1.distance(p2);  
          \this
```

知识讲解

调用distance方法时，除了传递p2参数外，还隐式的传递了p1作为参数，在方法中的this关键字即表示该参数。

- static修饰的方法则不需要针对某些对象进行操作，其运行结果仅仅与输入的参数有关，调用时直接用类名引用。

```
double c = Math.sqrt(3.0 * 3.0 + 4.0 * 4.0);
```

该方法在调用时，没有隐式的传递对象引用，因此在static方法中不可以使用this关键字。

static修饰方法 (续1)

• 由于static在调用时没有具体的对象，因此在static方法中不能对非static成员（对象成员）进行访问。static方法的作用在于提供一些“工具方法”和“工厂方法”等。

.....
Point.distance(Point p1, Point p2)
RandomUtils.nextInt()
StringUtils.leftPad(String str, int size, char padChar)
.....
Math.sqrt() Math.sin() Arrays.sort()

+

3.1.3. 【 static 关键字】 static 块

static块

• static块：属于类的代码块，在类加载期间执行的代码块，只执行一次，可以用来在软件中加载静态资源。

```
class Foo {
    static {
        //类加载期间，只执行一次
        System.out.println(" Load Foo.class ");
    }
    public Foo() {
        System.out.println("Foo()");
    }
}
Foo foo = new Foo();
```

输出的结果为：
 Load Foo.class
 Foo();
 Foo类加载时，运行静态块，在创建对象之前。

+

3.2. final 关键字

3.2.1. 【 final 关键字】 final 修饰变量

final修饰变量

• final关键字修饰成员变量，意为不可改变。
 • final修饰成员变量，两种方式初始化：
 – 声明同时初始化
 – 构造函数中初始化
 • final关键字也可以修饰局部变量，使用之前初始化即可。

```
public class Emp {
    private final int no = 100; // final变量声明时初始化
    public void testFinal() {
        no = 99;
    } //编译错误，final的变量不可被改变
}
```

+

3.2.2. 【final 关键字】 final 修饰方法

final修饰方法

- final关键字修饰的方法不可以被重写。
- 使一个方法不能被重写的意义在于：防止子类在定义新方法时造成的“不经意”重写。

知识讲解

```
public class Car {  
    // 点火  
    public void fire() {...}  
    ... ...  
}
```

```
public class Tank extends Car {  
    // 开炮  
    public void fire() {...}  
    ... ...  
}
```





+

3.2.3. 【final 关键字】 final 修饰类

Tarena
达内科技

final修饰类

- final关键字修饰的类不可以被继承。

```
final class Foo { }
```

```
class Goo extends Foo { } //会有编译错误
```

知识讲解

- JDK中的一些基础类库被定义为final的，例如：String、Math、Integer、Double 等等。
- 使一个类不能被继承的意义在于：可以保护类不被继承修改，可以控制滥用继承对系统造成的危害。

3.2.4. 【final 关键字】 static final 常量

Tarena
达内科技

static final常量

- static final 修饰的成员变量称为常量，必须声明同时初始化，不可被改变。
- static final 常量会在编译期被替换，例如：

知识讲解

```
class Foo {  
    public static final int NUM = 100;  
}  
  
class Goo {  
    public static void main(String[] args) {  
        System.out.println(Foo.NUM);  
        // 代码编译时，会替换为：System.out.println(100);  
    }  
}
```

static final 常量 Foo.NUM会在编译时被替换为其常量值（100），在运行Goo类时，Foo类不需要被载入。

经典案例

1. 重写 T 类和 J 类的 print 方法并测试

• 问题

在 TetrominoT 和 TetrominoJ 类中重写父类 Tetromino 的 print 方法，并进行测试，控制台输出结果如下所示：

· 方案

首先，在 TetrominoT 类中，重写 print 方法。方法的重写要遵循“两同两小一大”规则，“两同”即方法名相同，形参列表相同；“两小”指的是子类方法返回值类型应比父类方法返回值类型更小或相等，子类方法声明抛出的异常类应比父类方法声明抛出的异常类更小或相等；“一大”指的是子类方法的访问权限应比父类方法的访问权限更大或相等。其次，子类 TetrominoT 重写的 print 方法，首先实现在控制台输出“i am a T”，然后，使用 super 关键字调用父类的 print 方法。

然后，在 TetrominoJ 类中，在该类中重写 print 方法，仍然遵循上述原则。其次，子类 TetrominoJ 重写的 print 方法，首先实现在控制台输出“i am a J”，然后，使用 super 关键字调用父类的 print 方法。

最后，在 TetrominoGame 类中进行测试，查看调用方法的情况。

• 步骤

实现此案例需要按照如下步骤进行。

步骤一：在 TetrominoT 类中，重写 print 方法

首先，重构 TetrominoT 类，在该类中重写 print 方法。其次，子类 TetrominoT 重写的 print 方法，首先实现在控制台输出“i am a T”，然后，使用 super 关键字调用父类的 print 方法，代码如下所示：

```
public class TetrominoT extends Tetromino {  
    public TetrominoT(int row, int col) {  
        super();  
        // 按顺时针方向初始化 Cell  
        cells[0] = new Cell(row, col);  
        cells[1] = new Cell(row, col + 1);  
        cells[2] = new Cell(row, col + 2);  
        cells[3] = new Cell(row + 1, col + 1);  
    }  
  
    @Override  
    public void print() {  
        System.out.println("i am a T");  
        super.print();  
    }  
}
```

步骤二：在 TetrominoJ 类中，重写 print 方法

首先，重构 TetrominoJ 类，重写 print 方法；其次，子类 TetrominoJ 重写的 print 方法，首先实现在控制台输出“i am a J”，然后，使用 super 关键字调用父类的 print 方法。代码如下所示：

```
public class TetrominoJ extends Tetromino {  
    public TetrominoJ(int row, int col) {  
        // 按顺时针方向初始化 Cell  
        cells[0] = new Cell(row, col);  
    }  
}
```

```

        cells[1] = new Cell(row, col + 1);
        cells[2] = new Cell(row, col + 2);
        cells[3] = new Cell(row + 1, col + 2);
    }

    @Override
    public void print() {
        System.out.println("i am a J");
        super.print();
    }

}

```

步骤三：测试

重构 TetrominoGame 类，在其 main 方法中调用 print 方法。代码如下所示：

```

public class TetrominoGame {
    public static void main(String[] args) {
        //测试 TetrominoT
        System.out.println("-----打印 T型-----");
        Tetromino t = new TetrominoT(0, 4);

        t.print();

        printTetromino(t);

        //测试 TetrominoJ
        System.out.println("-----打印 J型-----");
        Tetromino j = new TetrominoJ(0, 4);

        j.print();

        printTetromino(j);
    }

    /**
     * 打印出游戏所在的平面（宽 10 格，高 20 格）。用“-”号表示平面上的每个单元，用“*”号打印
     * 显示方块中的每个格子
     *
     * @param tetromino 需要显示在游戏平面中的方块
     */
    public static void printTetromino(Tetromino tetromino) {
        int totalRow = 20;
        int totalCol = 10;
        //获取方块中存储的四个各自的数组
        Cell[] cells = tetromino.cells;
        for (int row = 0; row < totalRow; row++) {
            for (int col = 0; col < totalCol; col++) {
                // 用于判断该位置是否包含在 cells 数组中
                boolean isInCells = false;
                for (int i = 0; i < cells.length; i++) {
                    if (cells[i].row == row && cells[i].col == col) {
                        System.out.print("* ");
                        isInCells = true;
                    }
                }
                if (!isInCells) {
                    System.out.print("- ");
                }
            }
            System.out.println();
        }
    }
}

```

```
        break;
    }
}
if (!isInCells) {
    System.out.print("- ");
}
System.out.println();
}
```

控制台输出结果如下所示：

从以上输出结果可以看出，当把一个子类对象直接赋值给父类引用变量时，例如上面的代码：Tetromino t = new TetrominoT(0, 4);，这个 t 引用变量的编译时类型是

`Tetronimo`, 而运行时类型是 `TetrominoT`。当运行时, 如果子类覆盖了父类的某个方法 8 , 那么, 调用该引用变量的方法时, 其方法行为总是表现出子类方法的行为特征, 而不是父类方法的行为特征, 即, 调用子类的方法。上边的代码运行后, 输出了 “`i am a T`”、“`i am a J`” 也说明了调用了子类的 `print` 方法。

- 完整代码

本案例中, `Tetromino` 类的完整代码如下所示:

```
public class Tetromino {  
    Cell[] cells; // 属性, 用来存储一个方块的四个格子的坐标  
  
    /**  
     * 构造方法, 初始化 cells 数组  
     */  
    public Tetromino() {  
        cells = new Cell[4];  
    }  
    /**  
     * 按顺时针方向, 打印方块中四个格子所在的坐标  
     */  
    public void print() {  
        String str = "";  
        for (int i = 0; i < cells.length - 1; i++) {  
            str += "(" + cells[i].getCellInfo() + "), ";  
        }  
        str += "(" + cells[cells.length - 1].getCellInfo() + ")";  
        System.out.println(str);  
    }  
  
    /**  
     * 使方块下落一个格子  
     */  
    public void drop() {  
        for (int i = 0; i < cells.length; i++) {  
            cells[i].row++;  
        }  
    }  
  
    /**  
     * 使方块左移一个格子  
     */  
    public void moveLeft() {  
        for (int i = 0; i < cells.length; i++) {  
            cells[i].col--;  
        }  
    }  
  
    /**  
     * 使用方块右移一个格子  
     */  
    public void moveRight() {  
        for (int i = 0; i < cells.length; i++) {  
            cells[i].col++;  
        }  
    }  
}
```

`TetrominoT` 类的完整代码如下所示:

```
public class TetrominoT extends Tetromino {
    public TetrominoT(int row, int col) {
        super();
        // 按顺时针方向初始化 Cell
        cells[0] = new Cell(row, col);
        cells[1] = new Cell(row, col + 1);
        cells[2] = new Cell(row, col + 2);
        cells[3] = new Cell(row + 1, col + 1);
    }

    @Override
    public void print() {
        System.out.println("i am a T");
        super.print();
    }
}
```

TetrominoT 类的完整代码如下所示：

```
public class TetrominoJ extends Tetromino {
    public TetrominoJ(int row, int col) {
        // 按顺时针方向初始化 Cell
        cells[0] = new Cell(row, col);
        cells[1] = new Cell(row, col + 1);
        cells[2] = new Cell(row, col + 2);
        cells[3] = new Cell(row + 1, col + 2);
    }

    @Override
    public void print() {
        System.out.println("i am a J");
        super.print();
    }
}
```

TetrominoGame 类的完整代码如下所示：

```
public class TetrominoGame {
    public static void main(String[] args) {
        // 测试 TetrominoT
        System.out.println("-----打印 T 型-----");
        Tetromino t = new TetrominoT(0, 4);
        t.print();
        printTetromino(t);

        // 测试 TetrominoJ
        System.out.println("-----打印 J 型-----");
        Tetromino j = new TetrominoJ(0, 4);
        j.print();
        printTetromino(j);
    }

    /**
     * 打印出游戏所在的平面（宽 10 格，高 20 格）。用“—”号表示平面上的每个单元，用“*”号打印显示方块中的每个格子
     */
    * @param tetromino 需要显示在游戏平面中的方块
    */
    public static void printTetromino(Tetromino tetromino) {
        int totalRow = 20;
        int totalCol = 10;
```

```
//获取方块中存储的四个各自的数组
Cell[] cells = tetromino.cells;
for (int row = 0; row < totalRow; row++) {
    for (int col = 0; col < totalCol; col++) {
        // 用于判断该位置是否包含在 cells 数组中
        boolean isInCells = false;
        for (int i = 0; i < cells.length; i++) {
            if (cells[i].row == row && cells[i].col == col) {
                System.out.print("* ");
                isInCells = true;
                break;
            }
        }
        if (!isInCells) {
            System.out.print("- ");
        }
    }
    System.out.println();
}
```

达内IT培训

课后作业

1. 说出下面代码的输出结果，并解释原因

```
//鸵鸟
public class Ostrich extends Bird{
    public void fly(){
        System.out.println("我只能在地上奔跑...");
    }
    public static void main(String[] args) {
        Ostrich os=new Ostrich();
        os.fly();
    }
}
class Bird {
    public void fly(){
        System.out.println("我在天空里自由自在的飞翔...");
    }
}
```

2. 说出下面代码的输出结果，并解释原因

```
public class SlowPoint extends Point {
    public void move(int dx, int dy) {
        System.out.println("SlowPoint move parameter");
        move();
    }
    public static void main(String[] args) {
        SlowPoint sp=new SlowPoint();
        sp.move(10,20);
    }
}

class Point {
    public void move(int dx, int dy) {
        System.out.println("Point move parameter");
    }
    public void move(){
        System.out.println("Point move ");
    }
}
```

3. 关于 package 和 import 语句，下面说法错误的是：

- A. package 提供了一种命名机制，用于管理类名空间
- B. 定义类时，除了定义类的名称以外，必须要指定一个包名
- C. import 语句用于导入所需要的类
- D. 同时使用不同包中相同类名的类，包名不能省略

4. 关于 public 和 private，下面说法错误的是：

- A. private 修饰的成员变量和方法仅仅只能在本类中访问

- B. `public` 修饰的成员变量和方法可以在任何地方访问
- C. `private` 修饰的成员变量和方法可以在本类和子类中访问
- D. `public` 修饰的成员变量和方法只能在同一个包中访问

5. 关于 `protected` 关键字，下面说法错误的是：

- A. 用 `protected` 修饰的成员变量和方法可以被子类及同一个包中的类使用
- B. 使用 `protected` 关键字修饰的访问控制，即默认访问控制
- C. 默认访问控制的成员变量和方法可以被同一个包中的类访问
- D. 用 `protected` 修饰的成员变量和方法只能被子类使用

6. 关于 `static` 关键字，下面说法正确的是：

- A. 用 `static` 修饰的成员变量是属于对象的数据结构
- B. 在 `static` 方法中，可以访问非 `static` 成员（对象成员）
- C. `static` 成员变量存储在堆中
- D. 一个类的 `static` 成员变量只有“一份”，无论该类创建了多少对象

7. 关于 `final` 关键字，下面说法正确的是：

- A. `final` 关键字如果用于修饰成员变量，那么该成员变量必须在声明时初始化
- B. `final` 关键字修饰的类只能被继承一次
- C. `final` 关键字修饰的方法不可以被重写
- D. `final` 关键字如果用于修饰方法，该方法所在的类不能被继承

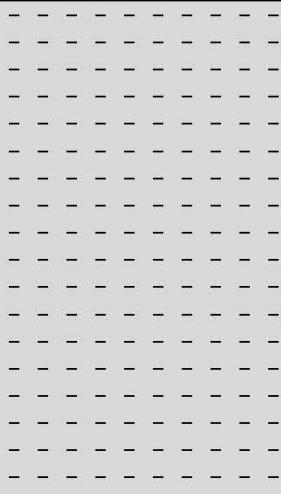
8. 关于声明静态常量，下面代码，正确的是：

- A. `public static String FOO = "foo";`
- B. `public static final String FOO = "foo";`
- C. `public final String FOO = "foo";`
- D. `public final static String FOO = "foo";`

9. 完成 TetrominoGame(提高题，选做)

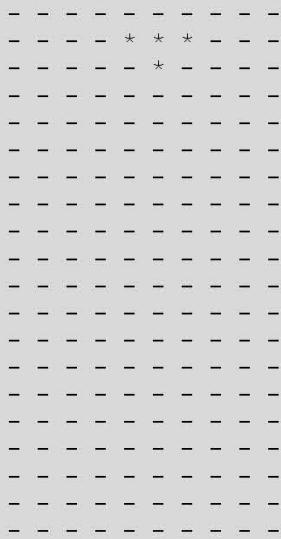
在课上案例“重写 T 类和 J 类的 `print` 方法并测试”的基础上，实现控制台版的对 T 型方块的下落，左移及右移，控制台输入效果如下所示：

```
-----打印 T 型-----
i am a T
(0,4), (0,5), (0,6), (1,5)
- - - - * * * - - -
- - - - - * - - - -
```



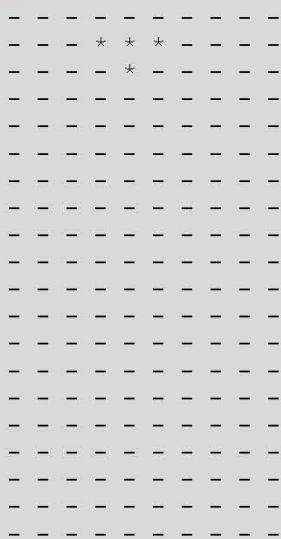
1 — 下落 , 2—向左 , 3—向右 , 0 — 退出

1



1 — 下落 , 2—向左 , 3—向右 , 0 — 退出

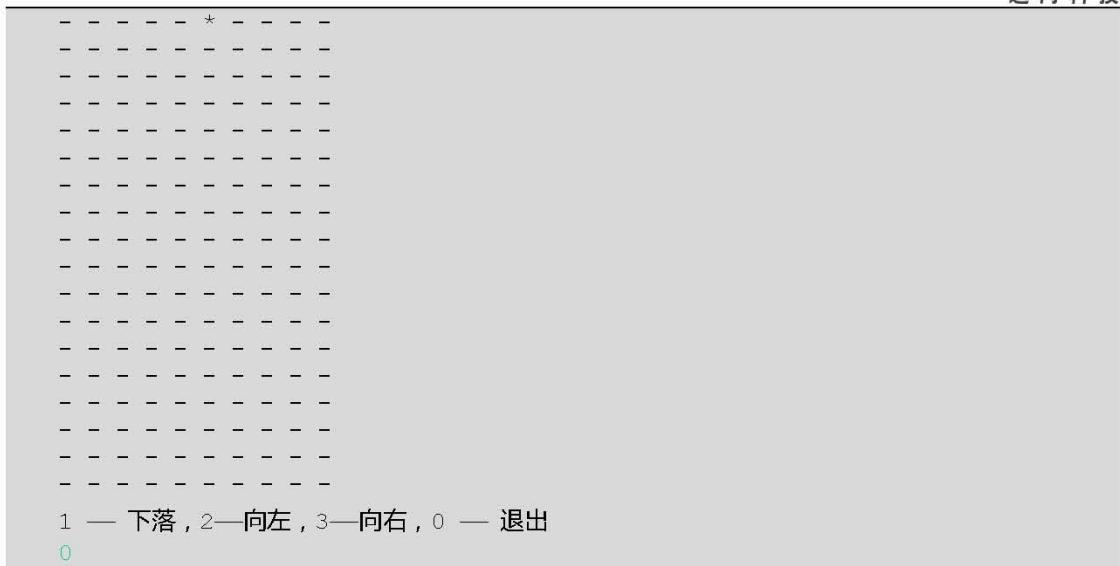
2



1 — 下落 , 2—向左 , 3—向右 , 0 — 退出

3





当用户选择 1 时，表示选择了下落功能；当用户选择 2 时，表示选择向左移动功能；当用户选择了 3 时，表示用户选择了向右移动功能；当用户选择 0 表示用户选择了退出功能。

Java 面向对象

Unit05

知识体系.....Page 99

抽象类和接口	使用抽象类	抽象方法和抽象类
		抽象类不可以实例化
		继承抽象类
		抽象类的意义
	使用接口	定义一个接口
		实现接口
		接口的继承
		接口和抽象类的区别

经典案例.....Page 102

根据周长计算不同形状图形的面积	抽象方法和抽象类
	抽象类不可以实例化
	继承抽象类
	抽象类的意义
银行卡系统 (实现银联接口)	定义一个接口
	实现接口
	接口的继承
	接口和抽象类的区别

课后作业.....Page 115

1. 抽象类和接口

1.1. 使用抽象类

1.1.1. 【使用抽象类】抽象方法和抽象类

Technology
Tarena
达内科技

抽象方法和抽象类

- 由abstract修饰的方法为抽象方法，抽象方法只有方法的定义，没有方法体实现，用一个分号结尾；
- 一个类中如果包含抽象方法，该类应该用abstract关键字声明为抽象类；
- 如果一个类继承了抽象类，必须重写其抽象方法（除非该类也声明为抽象类）。

知识讲解

```
abstract class Shape {  
    private double c;  
    public Shape(double c) {  
        this.c = c;  
    }  
    public abstract double area();  
}
```

1.1.2. 【使用抽象类】 抽象类不可以实例化

Technology
Tarena
达内科技

抽象类不可以实例化

- 抽象类不可以实例化，例如，如果Shape是抽象类的话，下面的代码是错误的：

```
Shape s1 = new Shape();
```

知识讲解

- 即使一个类中没有抽象方法，也可以将其定义为抽象类，同样，该类不可以实例化。
- abstract和final关键字不可以同时用于修饰一个类，因为final关键字使得类不可继承，而abstract修饰的类如果不可以继承将没有任何意义。

1.1.3. 【使用抽象类】继承抽象类

Technology
Tarena
达内科技

继承抽象类

- 一个类继承抽象类后，必须重写其抽象方法，不同的子类可以有不同的实现。

知识讲解

```
class Square extends Shape {  
    private double c;  
    public Square(double c) {  
        super(c);  
    }  
    public double area() {  
        return 0.0625*c*c;  
    }  
}
```

```
class Circle extends Shape {  
    private double c;  
    public Circle(double c) {  
        super(c);  
    }  
    public double area() {  
        return 0.0796*c*c;  
    }  
}
```

1.1.4. 【使用抽象类】抽象类的意义

抽象类的意义

- 抽象类的意义在于：
 - 为其子类提供一个公共的类型；
 - 封装子类中的重复内容（成员变量和方法）；
 - 定义有抽象方法，子类虽然有不同的实现，但该方法的定义是一致的。

知识讲解

+

1.2. 使用接口

1.2.1. 【使用接口】 定义一个接口

通过interface关键字定义接口

```
interface Runner {  
    public static int DEFAULT_SPEED = 100;  
    public void run();  
}
```

接口中不可以定义成员变量，但可以定义常量。

接口中只可以定义没有实现的方法（可以省略public abstract）。

1.2.2. 【使用接口】实现接口

Tarena
达内科技

实现接口

- 与继承不同，一个类可以实现多个接口，实现的接口直接用逗号分隔。当然，该类需要实现这些接口中定义的所有方法；
- 一个类可以通过`implements`关键字“实现”接口。一个类实现了某个接口后必须实现该接口中定义的所有方法。

知识讲解

```
class AmericanCurl implements Runner, ... {  
    public void run() {  
        System.out.println("run...");  
    }  
}
```

Technology
Tarena
达内科技

实现接口（续1）

- 接口可以作为一种类型声明变量，一个接口类型的变量可以引用实现了该接口的类的对象；通过该变量可以调用该接口中定义的方法（具体的实现类提供了方法的实现）。

Runner runner = new AmericanCurl();

知识讲解

1.2.3. 【使用接口】 接口的继承

接口的继承

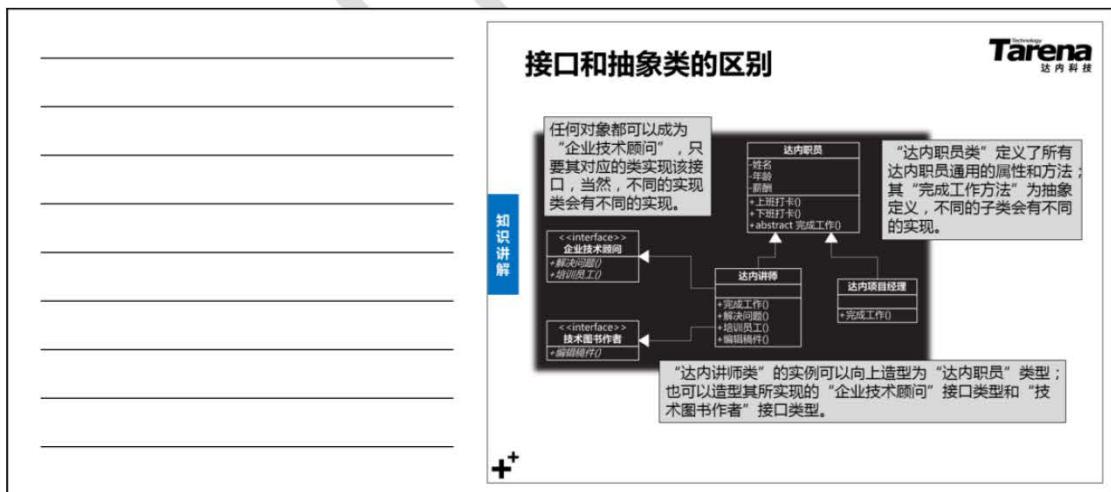
- 接口间可以存在继承关系，一个接口可以通过extends关键字继承另外一个接口。子接口继承了父接口中定义的所有方法。

知识讲解

```
interface Runner {  
    public void run();  
}  
  
interface Hunter  
    extends Runner {  
    public void hunt();  
}  
  
class AmericanCurl  
    implements Hunter {  
    public void run() { ... }  
    public void hunt() { ... }  
}
```

AmericanCurl必须实现Hunter接口中的hunt方法以及其父接口Runner中的run方法。

1.2.4. 【使用接口】 接口和抽象类的区别



经典案例

1. 根据周长计算不同形状图形的面积

• 问题

根据周长计算不同形状图形的面积，详细要求如下：

1. 计算多种图形的面积，并比较各种图形面积的最大值。
2. 使用抽象类及其子类的方式实现本案例。
3. 本案例以圆形和正方形为例。

注：正方形的面积公式为： $0.0625*c*c$ 。圆形的面积公式为： $0.0796*c*c$,其中，c 表示图形的周长。

• 方案

分析问题中的描述，可以得出如下解决方案：

1. 定义两个类 Square 和 Circle，分别表示正方形和圆形。
2. 正方形和圆形都有周长，我们可以使用 c 属性来表示。要计算正方形和圆形的面积，我们定义 area 方法来实现。即 分别在 Square 类和 Circle 类中定义 c 属性和 area 方法，并根据各自图形的公式计算对应的面积。
3. 案例问题中要求计算各种图形的面积，并找出最大值。在此，我们需要找到一种类型，该类型为 Square 类和 Circle 类的父类，使用该类型的数组来存储所有图形。因此，定义 Shape 类，该类为 Square 和 Circle 类的父类，并将 Square 类和 Circle 类的共有属性放入 Shape 类中定义。
4. 又因为父类(Shape)的引用不能直接调用子类(Square 或 Circle)的方法(area 方法)，因此，将 area 方法抽取到父类 Shape 中。在父类 Shape 中，不知道具体是哪种图形，因此 area 方法不知如何实现，在此，将该方法定义抽象方法，那么 Shape 类也要定义为抽象类。这样，也形成了子类(Square 或 Circle)重写父类(Shape)的 area 方法。
5. 最后，我们就可以定义如下方法来实现求面积的最大值：

```
public static void maxArea(Shape[] shapes) {  
    Shape s0 = shapes[0];  
    double max = s0.area();  
    int maxIndex = 0;  
    for (int i = 1; i < shapes.length; i++) {  
        double area = shapes[i].area();  
        if (area > max) {  
            max = area;  
            maxIndex = i;  
        }  
    }  
    System.out.println("数组中索引为"+maxIndex+"的图形的面积最大,面积为:"+max);  
}
```

以上方法中，参数 `shapes` 为 `Shape[]` 数组类型，该参数中可以存储正方形、圆形、或是其它图形。

• 步骤

实现此案例需要按照如下步骤进行。

步骤一：定义类 `Square` 和 `Circle`

定义两个类 `Square` 和 `Circle`，分别表示正方形和圆形，`Square` 类代码如下所示：

```
public class Square {  
}
```

`Circle` 类代码如下所示：

```
public class Circle {  
}
```

步骤二：定义 `c` 属性和 `area` 方法

分别在 `Square` 类和 `Circle` 类中定义 `c` 属性和 `area` 方法，并根据各自图形的公式计算对应的面积，`Square` 类代码如下所示：

```
public class Square {  
  
    private double c;  
    public Square(double c) {  
        this.c = c;  
    }  
    /**  
     * 计算正方形的面积  
     */  
    public double area() {  
        return 0.0625*c*c;  
    }  
}
```

`Circle` 类代码如下所示：

```
public class Circle {  
  
    private double c;  
    public Circle(double c) {  
        this.c = c;  
    }  
    /**  
     * 计算圆形的面积  
     */  
    public double area() {  
        return 0.0796*c*c;  
    }  
}
```

步骤三：定义父类 Shape 并抽取属性和方法

定义父类 Shape，将子类 Square 和 Circle 中共有属性 c 和方法 area 抽取到父类 Shape 中，并重构 Square 类和 Circle 类，Shape 类代码如下所示：

```
public abstract class Shape {  
    protected double c;  
    public abstract double area();  
}
```

Square 类代码如下所示：

```
public class Square extends Shape {  
  
    public Square(double c) {  
        this.c = c;  
    }  
    /**  
     * 计算正方形的面积  
     */  
    public double area() {  
        return 0.0625*c*c;  
    }  
}
```

Circle 类代码如下所示：

```
public class Circle extends Shape {  
  
    public Circle(double c) {  
        this.c = c;  
    }  
    /**  
     * 计算圆形的面积  
     */  
    public double area() {  
        return 0.0796*c*c;  
    }  
}
```

步骤四：计算各种图形面积的最大值

新建类 TestShape，在该类中新建方法 maxArea 方法，该方法实现计算多种图形面积的最大值，代码如下所示：

```
public class TestShape{  
    public static void maxArea(Shape[] shapes) {  
        double max = shapes[0].area();  
        int maxIndex = 0;  
        for (int i = 1; i < shapes.length; i++) {  
            double area = shapes[i].area();  
            if (area > max) {  
                max = area;  
                maxIndex = i;  
            }  
        }  
    }  
}
```

```
        System.out.println("数组中索引为"+maxIndex+"的图形的面积最大 面积为 :" +max);  
    }  
}
```

步骤五：测试

在 TestShape 类中，测试 maxArea 方法能否计算出各种图形面积的最大值，代码如下所示：

```
public class TestShape {  
  
    public static void main(String[] args) {  
        Shape[] shapes = new Shape[2];  
        shapes[0] = new Circle(4); //数组中的第一个元素为圆形对象，周长为 4  
        shapes[1] = new Square(4); //数组中的第二个元素为正方形对象，周长为 4  
        maxArea(shapes);  
    }  
  
    public static void maxArea(Shape[] shapes) {  
        double max = shapes[0].area();  
        int maxIndex = 0;  
        for (int i = 1; i < shapes.length; i++) {  
            double area = shapes[i].area();  
            if (area > max) {  
                max = area;  
                maxIndex = i;  
            }  
        }  
        System.out.println("数组中索引为" + maxIndex + "的图形的面积最大 ,面积为 :" + max);  
    }  
}
```

步骤六：运行

运行 TestShape 类，控制台输出结果如下所示：

```
数组中索引为 0 的图形的面积最大 ,面积为 :1.2736
```

观察上述输出结果，可以看出，shapes[0]表示的是圆形，周长相同的情况下，圆形的面积更大些。

- 完整代码

本案例中，Square 类的完整代码如下所示：

```
public class Square extends Shape {  
    public Square(double c) {  
        this.c = c;  
    }  
    /**  
     * 计算正方形的面积  
     */  
    public double area() {  
        return 0.0625*c*c;
```

```
}
```

Circle 类的完整代码如下所示：

```
public class Circle extends Shape {  
    public Circle(double c) {  
        this.c = c;  
    }  
    /**  
     * 计算圆形的面积  
     */  
    public double area() {  
        return 0.0796*c*c;  
    }  
}
```

Shape 类的完整代码如下所示：

```
public abstract class Shape {  
    protected double c;  
  
    public abstract double area();  
}
```

TestShape 类的完整代码如下所示：

```
public class TestShape {  
    public static void main(String[] args) {  
        Shape[] shapes = new Shape[2];  
        shapes[0] = new Circle(4); //数组中的第一个元素为圆形对象，周长为 4  
        shapes[1] = new Square(4); //数组中的第二个元素为正方形对象，周长为 4  
        maxArea(shapes);  
    }  
    public static void maxArea(Shape[] shapes) {  
        double max = shapes[0].area();  
        int maxIndex = 0;  
        for (int i = 1; i < shapes.length; i++) {  
            double area = shapes[i].area();  
            if (area > max) {  
                max = area;  
                maxIndex = i;  
            }  
        }  
        System.out.println("数组中索引为" + maxIndex + "的图形的面积最大，面积为：" + max);  
    }  
}
```

2. 银行卡系统（实现银联接口）

- 问题

本例要求实现银行卡系统的银联接口，详细要求如下：

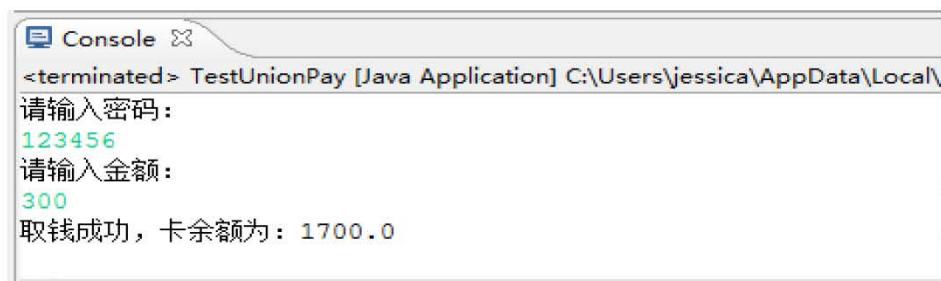
1. 银联接口，用于描述银联统一制定的规则，该接口提供检测密码方法、取钱方法以及查询余额方法。

2. 工商银行接口，用于描述工商银行发行的卡片功能，在满足银联接口的规则基础上，增加了在线支付功能。

3. 农业银行接口，用于描述中国农业银行发行的卡片功能，在满足银联接口的规则基础上，增加了支付电话费的功能。另外，农行的卡的卡内余额，允许最多透支 2000。

4. 实现工商银行接口和农业银行接口，并进行测试。

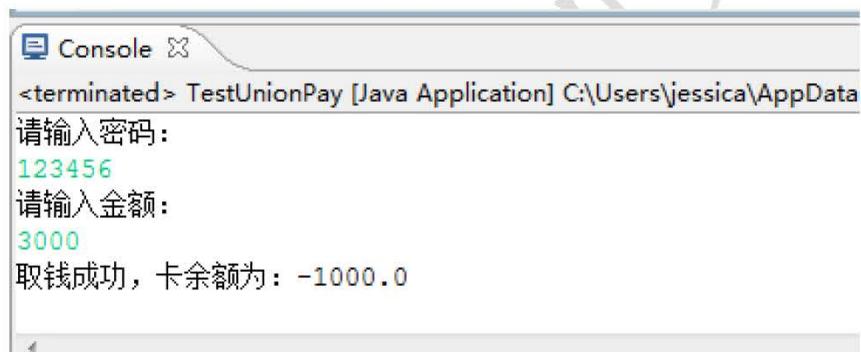
工行卡的控制台交互效果如图-1 所示：



```
Console <terminated> TestUnionPay [Java Application] C:\Users\jessica\AppData\Local\
请输入密码:
123456
请输入金额:
300
取钱成功，卡余额为： 1700.0
```

图-1

农行卡的控制台交互效果如图-2 所示：



```
Console <terminated> TestUnionPay [Java Application] C:\Users\jessica\AppData
请输入密码:
123456
请输入金额:
3000
取钱成功，卡余额为： -1000.0
```

图-2

从图-2 中可以看出，农行卡的卡内余额可以为负数。

• 步骤

实现此案例需要按照如下步骤进行。

步骤一：定义银联接口

首先，定义名为 UnionPay 的银联接口，用于描述银联统一制定的规则；然后，在该接口中定义 getBalance 方法表示获取余额功能、定义 drawMoney 表示取钱功能、定义 checkPwd 表示检查密码功能，代码如下所示：

```
/**
 * 接口：用于描述银联统一制定的规则
 */
public interface UnionPay {
    /**查看余额*/
    public double getBalance();
```

```

    /**
     * 取钱
     */
    public boolean drawMoney(double number);

    /**
     * 检查密码
     */
    public boolean checkPwd(String input);
}

```

步骤二：定义工商银行接口

定义名为 ICBC 的接口表示工商银行接口，用于描述中国工商银行发行的卡片功能，该接口需要满足银联接口的功能，因此，继承银联接口；该接口在具备银联接口的功能基础上，要求增加在线支付功能，所以，在该接口中定义 payOnline 方法，表示此功能，代码如下所示：

```

    /**
     * 接口：用于描述工商银行发行的卡片功能，在满足
     * 银联的规则基础上，添加自己特有的功能
     */
    public interface ICBC extends UnionPay {
        /**
         * 增加的在线支付功能
         */
        public void payOnline(double number);
    }

```

步骤三：定义农业银行接口

定义名为 ABC 的接口表示农业银行接口，用于描述中国农业银行发行的卡片功能，该接口需要满足银联接口的功能，因此，需要继承银联接口；该接口在具备银联接口的功能基础上，要求增加支付电话费功能，所以，在该接口中定义 payTelBill 方法，表示此功能，代码如下所示：

```

    /**
     * 接口：用于描述中国农业银行发行的卡片功能，在满足
     * 银联的规则基础上，添加自己特有的功能
     */
    public interface ABC extends UnionPay {
        /**
         * 增加支付电话费的功能
         */
        public boolean payTelBill(String phoneNum, double sum);
    }

```

步骤四：定义工商银行接口的实现类

首先，定义为名 ICBCImpl 的类，该类实现 ICBC 接口；另外，分析问题中的取钱功能，需要对余额信息进行存储，插入卡片以后需要输入密码后才能进行取钱，因此，在 ICBCImpl 类中定义 double 类型属性 money 表示账户余额以及 String 类型属性 pwd 表示卡片的密码，代码如下所示：

```

    /**
     * 类：用于描述工商银行实际发行的卡片
     * 该卡片具有的功能来自于继承的已经符合银联规范的 ICBC 接口
     */
    public class ICBCImpl implements ICBC {

```

```
private double money;
private String pwd;

public ICBCImpl(double money, String pwd) {
    this.money = money;
    this.pwd = pwd;
}
```

步骤五：实现检查密码、获取余额、取款、在线支付功能

在 ICBCImpl 类实现 checkPwd 方法、getBalance 方法、drawMoney 方法以及 payOnline 方法，代码如下所示：

```
/*
 * 类：用于描述工商银行实际发行的卡片
 * 该卡片具有的功能来自于继承的已经符合银联规范的 ICBC 接口
 */
public class ICBCImpl implements ICBC {
    private double money;
    private String pwd;

    public ICBCImpl(double money, String pwd) {
        this.money = money;
        this.pwd = pwd;
    }

    @Override
    public double getBalance() {
        return money;
    }

    @Override
    public boolean drawMoney(double number) {
        if(number <= money){
            money -= number;
            return true;
        }
        return false;
    }

    @Override
    public void payOnline(double number) {
        if(number < money){
            money -= number;
        }
    }

    @Override
    public boolean checkPwd(String input) {
        if(pwd.equals(input))
            return true;
        else
            return false;
    }
}
```

以上四个方法的实现逻辑为：

取钱功能的实现为在当前余额的基础上减去要取的金额，如 drawMoney 方法的实现。

在线付款功能的实现为当前余额的基础上减去要支付的金额，如 payOnline 方法的实现。

检查密码功能的实现为检查卡的密码与用户输入的密码是否相等，如果相等返回 true，否则返回 false，如 checkPwd 方法的实现。

获取余额功能的实现为类 ICBCImpl，类中 money 属性即表示余额，将其返回即可，如 getBalance 方法的实现。

步骤六：定义农业银行接口的实现类

实现农业银行接口的过程与实现工行接口的过程类似，实现的过程中注意，农行卡的卡内余额可以透支 2000，代码如下所示：

```
/*
 * 类：用于描述农业银行实际发行的卡片
 * 该卡片具有的功能来自于继承的已经符合银联规范的 ABC 接口
 */
public class ABCImpl implements ABC {
    /**卡内余额，允许最多透支 2000*/
    private double balance;
    /**账号密码*/
    private String password;

    public ABCImpl(double balance, String password) {
        this.balance = balance;
        this.password = password;
    }

    @Override
    public double getBalance() {
        return balance;
    }

    @Override
    public boolean drawMoney(double number) {
        if((balance-number) >= -2000){
            balance-=number;
            return true;
        }
        return false;
    }

    @Override
    public boolean checkPwd(String input) {
        if(password.equals(input)){
            return true;
        }else{
            return false;
        }
    }

    @Override
    public boolean payTelBill(String phoneNum, double sum) {
        if(phoneNum.length() == 11 && (balance-sum)>=-2000){
            balance-=sum;
            return true;
        }
    }
}
```

```
        }
        return false;
    }
}
```

步骤七：测试

新建 TestUnionPay 类，测试银联接口提供的方法，是否成功实现，代码如下所示：

```
import java.util.Scanner;
/**
 * 测试实现接口后的类的方法调用
 */
public class TestUnionPay {

    public static void main(String[] args) {
        UnionPay icbc = new ICBCImpl(2000,"123456");

        Scanner input = new Scanner(System.in);
        System.out.println("请输入密码：");
        if(icbc.checkPwd(input.next())){
            System.out.println("请输入金额：");
            double num = Double.parseDouble(input.next());
            if(icbc.drawMoney(num)){
                System.out.println("取钱成功，卡余额为："+icbc.getBalance());
            }
            else{
                System.out.println("取钱失败");
            }
        }
        else{
            System.out.println("密码错误");
        }
        input.close();
    }
}
```

查看上述代码，可以看出以上代码是采用 ICBCImpl 类来实例化银联接口 UnionPay 的，当然，也可以使用 ABCImpl 来实例化银联接口 UnionPay，此时，卡内余额是可以透支 2000 元的。

• 完整代码

本案例中，UnionPay 接口的完整代码如下所示：

```
/**
 * 接口：用于描述银联统一制定的规则
 */
public interface UnionPay {
    /**查看余额*/
    public double getBalance();
    /**取钱*/
    public boolean drawMoney(double number);
    /**检查密码*/
    public boolean checkPwd(String input);
}
```

ICBC 接口的完整代码如下所示：

```
/**  
 * 接口：用于描述工商银行发行的卡片功能，在满足  
 * 银联的规则基础上，添加自己特有的功能  
 */  
public interface ICBC extends UnionPay {  
    /**增加的在线支付功能*/  
    public void payOnline(double number);  
}
```

ABC 接口的完整代码如下所示：

```
/**  
 * 接口：用于描述中国农业银行发行的卡片功能，在满足  
 * 银联的规则基础上，添加自己特有的功能  
 */  
public interface ABC extends UnionPay {  
    /**增加支付电话费的功能*/  
    public boolean payTelBill(String phoneNum,double sum);  
}
```

ICBCImpl 类的完整代码如下所示：

```
/**  
 * 类：用于描述工商银行实际发行的卡片  
 * 该卡片具有的功能来自于继承的已经符合银联规范的 ICBC 接口  
 */  
public class ICBCImpl implements ICBC {  
    private double money;  
    private String pwd;  
  
    public ICBCImpl(double money,String pwd){  
        this.money = money;  
        this.pwd = pwd;  
    }  
  
    @Override  
    public double getBalance() {  
        return money;  
    }  
  
    @Override  
    public boolean drawMoney(double number) {  
        if(number <= money){  
            money -=number;  
            return true;  
        }  
        return false;  
    }  
  
    @Override  
    public void payOnline(double number) {  
        if(number < money){  
            money-=number;  
        }  
    }  
  
    @Override  
    public boolean checkPwd(String input) {
```

```
        if(pwd.equals(input))
            return true;
        else
            return false;
    }
}
```

ABCImpl 类的完整代码如下所示：

```
/**
 * 类：用于描述农业银行实际发行的卡片
 * 该卡片具有的功能来自于继承的已经符合银联规范的 ABC 接口
 */
public class ABCImpl implements ABC {
    /**卡内余额，允许最多透支 2000*/
    private double balance;
    /**账号密码*/
    private String password;

    public ABCImpl(double balance, String password){
        this.balance = balance;
        this.password = password;
    }

    @Override
    public double getBalance() {
        return balance;
    }

    @Override
    public boolean drawMoney(double number) {
        if((balance-number) >= -2000){
            balance-=number;
            return true;
        }
        return false;
    }

    @Override
    public boolean checkPwd(String input) {
        if(password.equals(input)){
            return true;
        }else{
            return false;
        }
    }

    @Override
    public boolean payTelBill(String phoneNum, double sum) {
        if(phoneNum.length() == 11 && (balance-sum)>=-2000){
            balance-=sum;
            return true;
        }
        return false;
    }
}
```

TestUnionPay 类的完整代码如下所示：

```
import java.util.Scanner;
/**
```

```
* 测试实现接口后的类的方法调用
*/
public class TestUnionPay {

    public static void main(String[] args) {
        UnionPay icbc = new ICBCImpl(2000,"123456");

        Scanner input = new Scanner(System.in);
        System.out.println("请输入密码: ");
        if(icbc.checkPwd(input.next())){
            System.out.println("请输入金额: ");
            double num = Double.parseDouble(input.next());
            if(icbc.drawMoney(num)){
                System.out.println("取钱成功, 卡余额为: "+icbc.getBalance());
            }
            else{
                System.out.println("取钱失败");
            }
        }
        else{
            System.out.println("密码错误");
        }
        input.close();
    }
}
```

课后作业

1. 查看下列代码

```
abstract class Vehicle {  
    public int speed() {  
        return 0;  
    }  
}  
  
class Car extends Vehicle {  
    public int speed() {  
        return 60;  
    }  
}  
  
class RaceCar extends Car {  
    public int speed() {  
        return 150;  
    }  
}  
  
public class TestCar {  
    public static void main(String[] args) {  
        RaceCar racer = new RaceCar();  
        Car car = new RaceCar();  
        Vehicle vehicle = new RaceCar();  
        System.out.println(racer.speed() + ", " + car.speed() + ", "  
        + vehicle.speed());  
    }  
}
```

上述代码运行的结果是

- A. 0,0,0
- B. 150,60,0
- C. 150,150,150
- D. 抛出运行时异常

2. 编写正六边形类继承 Shape , 实现其 area 方法

在课上案例“根据周长计算不同形状图形的面积”基础上 编写正六边形类继承 Shape , 实现其 area 方法。

注：正六边形的面积计算公式为： $0.0721 * c * c$ ，其中 c 为周长。

3. 简述抽象类的意义

4. 编写建设银行接口 CCB 继承银联接口，并实现该接口

在课上案例“银行卡系统（实现银联接口）”的基础上，编写建设银行接口 CCB。建设银行接口，用于描述中国建设银行发行的卡片功能，在满足银联接口的规则基础上，增加了

支付燃气费的功能。

5. 关于接口和抽象类，下列说法正确的是

- A. 抽象类和接口都不能实例化。
- B. 接口里只能包含抽象方法，抽象类则可以包含普通方法。
- C. 接口里不包含构造器，抽象类里可以包含构造器。
- D. 一个类只能实现一个接口

Java 面向对象

Unit06

知识体系.....Page 118

抽象类和接口	多态	多态的意义
		向上造型
		强制转型
		instanceof 关键字
内部类		定义成员内部类
		创建内部类对象
		定义匿名内部类
面向对象汇总		面向对象汇总

经典案例.....Page 121

ATM 机系统	多态的意义
	向上造型
	强制转型
	instanceof 关键字

课后作业.....Page 130

1. 抽象类和接口

1.1. 多态

1.1.1. 【多态】多态的意义

Technology
Tarena
达内科技

多态的意义

- 一个类型的引用在指向不同的对象时会有不同的实现：

```
达内职员 emp1 = new 达内讲师();  
达内职员 emp2 = new 达内项目经理();  
emp1.完成工作();  
emp2.完成工作();
```

知识点讲解

- 同样一个对象，造型成不同的类型时，会有不同的功能

```
达内讲师 teacher = new 达内讲师();  
企业技术顾问 consultant = teacher;  
技术图书作者 author = teacher;  
consultant.培训员工();  
author.编辑稿件();
```

1.1.2. 【多态】向上造型

达内讲师 teacher = new 达内讲师();
达内职员 emp = teacher;
企业技术顾问 consultant = teacher;
技术图书作者 author = teacher;

emp.上班打卡();
emp.完成工作();
consultant.培训员工();
author.编辑稿件();

```
classDiagram
    class Teacher {
        +上班打卡()
        +完成工作()
        +培训员工()
        +编辑稿件()
    }
    class Employee {
        +上班打卡()
        +完成工作()
    }
    class Consultant {
        +培训员工()
    }
    class Author {
        +编辑稿件()
    }
    Employee <|-- Teacher
    Consultant <|-- Teacher
    Author <|-- Teacher
```

达内讲师
达内职员
企业技术顾问
技术图书作者

上班打卡()
完成工作()
培训员工()
编辑稿件()

1.1.3. 【多态】强制转型

Technology
Tarena
达内科技

强制转型

- 可以通过强制转换将父类型变量转换为子类型变量，前提是该变量指向的对象确实是孩子类类型。
- 也可通过强制转换将变量转换为某种接口类型，前提是该变量指向的对象确实实现了该接口。
- 如果在强制转换过程中出现违背上述两个前提，将会抛出ClassCastException

知识讲解

```
达内职员 emp1 = new 达内讲师();
```

```
达内讲师 teacher = (达内讲师) emp1;  
技术图书作者 author = (技术图书作者) emp1;
```

```
达内项目经理 pm = (达内项目经理) emp1;  
//会抛出ClassCastException
```

1.1.4. 【多态】 instanceof 关键字

Technology
Tarena
达内科技

instanceof关键字

- 在强制转型中，为了避免出现ClassCastException，可以通过instanceof关键字判断某个引用指向的对象是否为指定类型。

知识点讲解

达内职员 emp1 = new 达内讲师();

通过instanceof运算判断
emp1引用所指向的对象是否
是指定类型。

```
System.out.println(emp1 instanceof 达内讲师); // true  
System.out.println(emp1 instanceof 技术图书作者); // true  
System.out.println(emp1 instanceof 达内项目经理); // false
```

```
达内项目经理 pm = null;  
if (emp1 instanceof 达内项目经理) {  
    pm = (达内项目经理) emp1;  
}
```

类型强制转换时最好配合
instanceof判断，这样可以避
免出现ClassCastException。

1.2. 内部类

1.2.1. 【内部类】定义成员内部类

Tarena
达内科技

定义成员内部类

• 一个类可以定义在另外一个类的内部，定义在类内部的类称之为Inner，其所在的类称之为Outer；

• Inner定义在Outer的内部。通常只服务于Outer，对外部不具备可见性，Inner可以直接调用Outer的成员及方法（包括私有的）。

知识讲解

```
class Outer {  
    private int time;  
  
    class Inner {  
        public void timeInc() {  
            time++;  
        }  
    }  
}
```

1.2.2. 【内部类】创建内部类对象

Technology
Tarena
达内科技

创建内部类对象

- 一般情况下，Inner对象会在Outer对象中创建（构造方法或其他方法）；Inner对象中会有一个隐式的引用指向创建它的Outer类对象。

知识讲解

A diagram illustrating the relationship between an Inner class and an Outer class. On the left, there is a rectangular box labeled "Inner". On the right, there is a larger rectangular box labeled "Outer". A curved arrow originates from the bottom edge of the "Inner" box and points to the top edge of the "Outer" box, representing an implicit reference.

```
Outer outer = new Outer(100);
outer.printTime();
```

上面代码的输出结果为：101。在Outer构造方法中创建的Inner对象会有一个隐式的引用指向创建它的Outer对象，调用Inner对象的timeInc方法，即会对Outer的time属性进行操作。

```
class Outer {
    private int time;
    private Inner inner;

    Outer(int time) {
        this.time = time;
        inner = new Inner();
        inner.timeInc();
    }

    public void printTime() {
        System.out.println(time);
    }
}

class Inner {
    public void timeInc() {
        time++;
    }
}
```

1.2.3. 【内部类】 定义匿名内部类

如果在一段程序中需要创建一个类的对象（通常这个类需要实现某个接口或者继承某个类），而且对象创建后，这个类的价值也就不存在了，这个类可以不必命名，称之为匿名内部类。

知识讲解

用匿名类所实现的接口或所继承的父类类型声明的引用

匿名类所要实现的接口或继承的父类

构造方法参数

匿名中定义的成员变量或方法

```
SuperType obj = new SuperType (...) {  
    ...  
};  
+
```

定义匿名内部类 (续1)

```
public interface Action {  
    public void execute();  
}  
  
public class Main {  
    public static void main(String[] args) {  
        Action action = new Action(){  
            public void execute() {  
                System.out.println("Hello, World");  
            }  
        };  
        action.execute();  
    }  
}
```

知识讲解

代码中创建了实现了Action接口的内部匿名类对象，并调用其execute方法。

1.3. 面向对象汇总

1.3.1. 【面向对象汇总】面向对象汇总

面向对象特征：

- 封装
- 继承
- 多态

知识讲解

+

经典案例

1. ATM 机系统

· 问题

在案例“银行卡系统（实现银联接口）”基础上实现本案例，本案例要求实现中国农业银行的ATM系统，详细要求如下：

1. 中国农业银行的 ATM 机对所有银联卡提供检查密码功能、取款功能以及查询余额功能。
 2. 如果为农行的卡，可以实现支付电话费功能。界面交互效果如下：
如果为农行的银行卡界面交互效果如图-1 所示。

如果为农行的银行卡界面交互效果如图-1 所示。

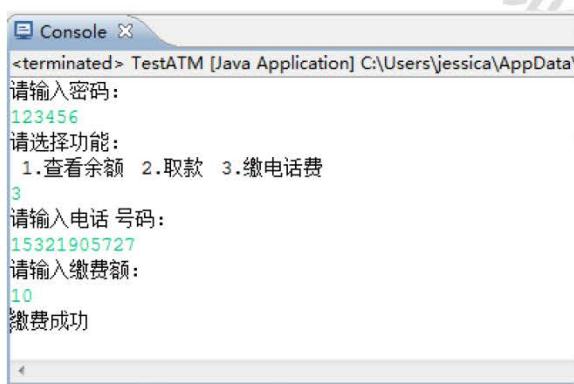


图-1

观察图-1 可以发现，农业银行的卡可以实现缴费功能。如果为其它银行的银联卡，界面交互效果如图-2 所示。



- 2

观察图-2 可以发现，工商银行的卡不能实现缴费功能。

· 方案

要实现农行的 ATM 机系统，解决方案如下：

1. 定义名为 ABCATM 的类，表示农行的 ATM 机系统。
 2. 在 ABCATM 类中，定义属性 card，该属性的类型为 UnionPay，表示不同银行的银

联卡。

3. 在 ABCATM 类中，定义方法 `insertCard`，实现向 ATM 机插入银联卡的操作。
4. 在 ABCATM 类中，定义方法 `outCard`，实现从 ATM 机取出银联卡的操作。
5. 在 ABCATM 类中，定义方法 `showBalance`，实现查询余额功能，在该方法中调用银联接口 UnionPay 的 `getBalance` 方法即可。
6. 在 ABCATM 类中，定义 `takeMoney` 方法，实现取款功能，在该方法中调用银联接口 UnionPay 的 `drawMoney` 方法即可。
7. 在 ABCATM 类中，定义 `payTelBill` 方法，实现支付电话费功能，在该方法中，首先判断插入的银联卡是否为农行的卡，如果是农行的卡，则调用农行接口 ABC 的 `payTelBill` 方法；否则，提示信息“您的卡不是农业银行卡，无法完成缴费”。
8. 在 ABCATM 类中，定义 `allMenu` 方法，实现农行 ATM 机的菜单。
9. 构建测试类，测试农行 ATM 机所实现的功能。

• 步骤

实现此案例需要按照如下步骤进行。

步骤一：定义类 ABCATM

定义名为 ABCATM 的类，表示农行的 ATM 机系统，代码如下所示：

```
/**  
 * ABCATM 机类，用于描述模拟插入银行卡后的操作  
 */  
public class ABCATM {  
}
```

步骤二：定义属性 card 表示银联卡

在 ABCATM 类中，定义属性 `card`，该属性的类型为 `UnionPay`，表示银联卡。代码如下所示：

```
/**  
 * ABCATM 机类，用于描述模拟插入银行卡后的操作  
 */  
public class ABCATM {  
  
    private UnionPay card;  
  
}
```

步骤三：实现 ATM 机插口操作

在 ABCATM 类中，定义方法 `insertCard`，实现向 ATM 机插入银联卡的操作。在该方法中，将 `card` 进行初始化，即表示插卡操作，代码如下所示：

```
import java.util.Scanner;
```

```
/**  
 * ABCATM 机类，用于描述模拟插入银行卡后的操作  
 */  
public class ABCATM {  
    private UnionPay card;  
  
    public void insertCard(UnionPay userCard){  
        if (card == null) {  
            card = userCard;  
        }  
    }  
}
```

步骤四：实现取卡操作

在 ABCATM 类中，定义方法 outCard，实现从 ATM 机取出银联卡的操作。在该方法中，将 card 的值设置为 null，即表示取卡操作，代码如下所示：

```
/**  
 * ABCATM 机类，用于描述模拟插入银行卡后的操作  
 */  
public class ABCATM {  
    private UnionPay card;  
  
    public void insertCard(UnionPay userCard){  
        if (card == null) {  
            card = userCard;  
        }  
    }  
  
    private void outCard(){  
        card = null;  
    }  
}
```

步骤五：实现查询余额功能

在 ABCATM 类中，定义 showBalance 方法实现查询余额功能，在该方法中调用银联接口 UnionPay 的 getBalance 方法即可，代码如下所示：

```
/**  
 * ABCATM 机类，用于描述模拟插入银行卡后的操作  
 */  
public class ABCATM {  
    private UnionPay card;  
  
    public void insertCard(UnionPay userCard){  
        if (card == null) {  
            card = userCard;  
        }  
    }  
  
    private void outCard(){  
        card = null;  
    }  
}
```

```
private void showBalance(){
    System.out.println("当前余额是：" + card.getBalance());
}

}
```

步骤六：实现取款功能

在 ABCATM 类中，定义 takeMoney 方法实现取款功能，在该方法中调用银联接口 UnionPay 的 drawMoney 方法即可，代码如下所示：

```
import java.util.Scanner;

/**
 * ABCATM 机类，用于描述模拟插入银行卡后的操作
 */
public class ABCATM {
    private UnionPay card;

    public void insertCard(UnionPay userCard) {
        if (card == null) {
            card = userCard;
        }
    }

    private void outCard(){
        card = null;
    }

    private void showBalance(){
        System.out.println("当前余额是：" + card.getBalance());
    }

    private void takeMoney(){
        Scanner input = new Scanner(System.in);
        System.out.println("请输入取款数目：");
        double number = Double.parseDouble(input.next());
        if(card.drawMoney(number)){
            System.out.println("取款成功");
        }else{
            System.out.println("取款失败");
        }
    }
}
```

步骤七：实现支付电话费功能

在 ABCATM 类中，定义 payTelBill 方法实现支付电话费功能，在该方法中，首先判断插入的银联卡是否为农行的卡，如果是农行的卡，则调用农行接口 ABC 的 payTelBill 方法；否则，提示信息“您的卡不是农业银行卡，无法完成缴费”，代码如下所示：

```

import java.util.Scanner;

/**
 * ABCATM 机类，用于描述模拟插入银行卡后的操作
 */
public class ABCATM {
    private UnionPay card;

    public void insertCard(UnionPay userCard) {
        if (card == null) {
            card = userCard;
        }
    }

    private void outCard(){
        card = null;
    }

    private void showBalance(){
        System.out.println("当前余额是：" + card.getBalance());
    }

    private void takeMoney(){
        Scanner input = new Scanner(System.in);
        System.out.println("请输入取款数目：");
        double number = Double.parseDouble(input.next());
        if(card.drawMoney(number)){
            System.out.println("取款成功");
        }else{
            System.out.println("取款失败");
        }
    }

    private void payTelBill(){
        Scanner input = new Scanner(System.in);
        if(card instanceof ABC){
            ABC abcCard = (ABC)card;
            System.out.println("请输入电话号码：");
            String telNum = input.next();
            System.out.println("请输入缴费额：");
            double sum = Double.parseDouble(input.next());
            if(abcCard.payTelBill(telNum,sum)){
                System.out.println("缴费成功");
            }else{
                System.out.println("缴费失败");
            }
        }else{
            System.out.println("您的卡不是农业银行卡，无法完成缴费");
        }
    }
}

```

步骤八：实现 ATM 机的菜单功能

在 ABCATM 类中，定义 allMenu 方法实现农行 ATM 机的菜单功能，代码如下所示：

```

import java.util.Scanner;

/**
 * ABCATM 机类，用于描述模拟插入银行卡后的操作
 */
public class ABCATM {
    private UnionPay card;

    public void insertCard(UnionPay userCard) {
        if (card == null) {
            card = userCard;
        }
    }

    private void outCard(){
        card = null;
    }

    private void showBalance(){
        System.out.println("当前余额是：" + card.getBalance());
    }

    private void takeMoney(){
        Scanner input = new Scanner(System.in);
        System.out.println("请输入取款数目：");
        double number = Double.parseDouble(input.next());
        if(card.drawMoney(number)){
            System.out.println("取款成功");
        }else{
            System.out.println("取款失败");
        }
    }

    private void payTelBill(){
        Scanner input = new Scanner(System.in);
        if(card instanceof ABC){
            ABC abcCard = (ABC)card;
            System.out.println("请输入电话 号码：");
            String telNum = input.next();
            System.out.println("请输入缴费额：");
            double sum = Double.parseDouble(input.next());
            if(abcCard.payTelBill(telNum,sum)){
                System.out.println("缴费成功");
            }else{
                System.out.println("缴费失败");
            }
        }else{
            System.out.println("您的卡不是农业银行卡，无法完成缴费");
        }
    }

    public void allMenu(){
        Scanner input = new Scanner(System.in);

        System.out.println("请输入密码：");
        String pwd = input.next();

        if(card.checkPwd(pwd)){

```

```

        System.out.println("请选择功能 :\n 1. 查看余额 2. 取款 3. 缴电话费");
        int choice = Integer.parseInt(input.next());
        switch(choice){
            case 1:
                showBalance();
                break;
            case 2:
                takeMoney();
                break;
            case 3:
                payTelBill();
                break;
            default:
                System.out.println("非法输入");
        }
    }else{
        System.out.print("密码错误");
    }
    this.outCard();
}

}

```

步骤九：测试

构建测试类 TestATM，测试农行 ATM 机所实现的功能，代码如下所示：

```

/**
 * 测试接口多态
 */
public class TestATM {
    public static void main(String[] args) {
        ABCATM atm =new ABCATM();
        //ICBCImpl icbc = new ICBCImpl(3000, "123456");//工商银行的卡
        ABCImpl abc = new ABCImpl(1000, "123456");//农业银行的卡

        //atm.insertCard(icbc);

        atm.insertCard(abc);

        atm.allMenu();
    }
}

```

运行上述代码后，会发现农行的卡可以实现支付电话费功能，而工商银行的卡则不能实现支付电话费功能。

- **完整代码**

本案例中，ABCATM 类的完整代码如下所示：

```

import java.util.Scanner;
/**/

```

```

 * ABCATM 机类，用于描述模拟插入银行卡后的操作
 */
public class ABCATM {
    private UnionPay card;

    public void insertCard(UnionPay userCard) {
        if (card == null) {
            card = userCard;
        }
    }

    private void outCard() {
        card = null;
    }

    private void showBalance() {
        System.out.println("当前余额是: " + card.getBalance());
    }

    private void takeMoney() {
        Scanner input = new Scanner(System.in);
        System.out.println("请输入取款数目: ");
        double number = Double.parseDouble(input.next());
        if (card.drawMoney(number)) {
            System.out.println("取款成功");
        } else {
            System.out.println("取款失败");
        }
    }

    private void payTelBill() {
        Scanner input = new Scanner(System.in);
        if (card instanceof ABC) {
            ABC abcCard = (ABC) card;
            System.out.println("请输入电话 号码: ");
            String telNum = input.next();
            System.out.println("请输入缴费额: ");
            double sum = Double.parseDouble(input.next());
            if (abcCard.payTelBill(telNum, sum)) {
                System.out.println("缴费成功");
            } else {
                System.out.println("缴费失败");
            }
        } else {
            System.out.println("您的卡不是农业银行卡，无法完成缴费");
        }
    }

    public void allMenu() {
        Scanner input = new Scanner(System.in);

        System.out.println("请输入密码: ");
        String pwd = input.next();

        if (card.checkPwd(pwd)) {
            System.out.println("请选择功能: \n 1.查看余额 2.取款 3.缴电话费");
            int choice = Integer.parseInt(input.next());
            switch (choice) {
                case 1:
                    showBalance();
                    break;
                case 2:
                    takeMoney();
                    break;
                case 3:
            }
        }
    }
}

```

```
        payTelBill();
        break;
    default:
        System.out.println("非法输入");
    }
} else {
    System.out.print("密码错误");
}
this.outCard();
}

}
```

TestATM类完整代码如下所示：

```
/**
 * 测试接口多态
 */
public class TestATM {
    public static void main(String[] args) {
        ABCATM atm =new ABCATM();
        //ICBCImpl icbc = new ICBCImpl(3000, "123456");//工商银行的卡
        ABCImpl abc = new ABCImpl(1000, "123456");//农行的卡

        //atm.insertCard(icbc);
        atm.insertCard(abc);
        atm.allMenu();
    }
}
```

课后作业

1. 请看下列代码：

```
public class SuperClass {  
    public static void main(String[] args) {  
        SuperClass sup=new SuperClass();  
        SubClass sub=new SubClass();  
        SuperClass supSub=new SubClass();  
    }  
}  
class SubClass extends SuperClass{  
}
```

根据上述代码，判断下列选项中，赋值语句错误的是：

- A. sup=sub;
- B. sub=sup;
- C. supSub=sub;
- D. supSub=sup;

2. 射击游戏

现有一个射击游戏中的场景如图- 1 所示。在图- 1 中有飞机、蜜蜂，这两种飞行物都可以被炮弹击中。如果击中的是敌人（即飞机），则获得 5 分；如果击中的是蜜蜂则获得奖励。本案例的详细需求如下：

1. 奖励有两种类型，一种奖励为双倍火力、另一种奖励为手弹。
2. 空中现有 3 架飞机、两只蜜蜂，判断某一炮弾能否击中它们。

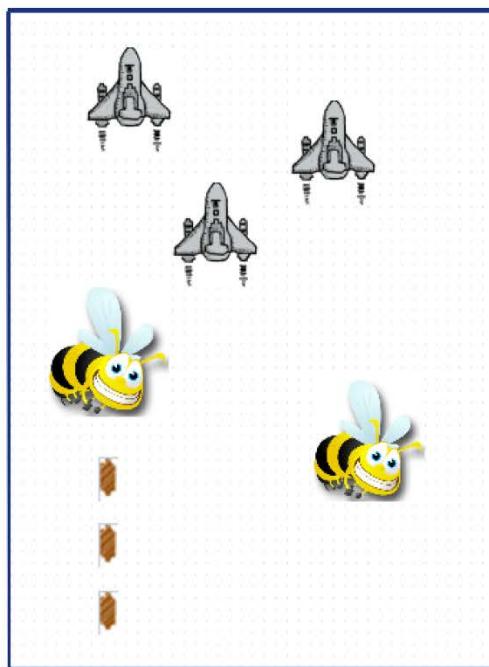


图-1