

# Java语言程序设计

配套教材由清华大学出版社出版发行

## 第3章 面向对象程序设计之一



中國農業大學

阚道宏

# 第3章 面向对象程序设计之一

- 程序=数据 + 算法
- 程序设计方法
  - 结构化程序设计
  - 面向对象程序设计
  - 代码重用 (code reuse)
- C语言：结构化程序设计语言
- Java语言：是一种面向对象程序设计语言



# 第3章 面向对象程序设计之一

- 本章内容
  - [3.1 面向对象程序设计方法](#)
  - [3.2 面向对象程序的设计过程](#)
  - [3.3 类与对象的语法细则](#)
  - [3.4 数组](#)
  - [3.5 Java程序文件的组织](#)



# 3.1 面向对象程序设计方法

- 程序是用于处理数据的，通常包括4项功能：
  - 定义保存数据的变量
  - 输入原始数据
  - 处理数据
  - 输出处理结果
- **程序实例：**编写一个计算长方形面积和周长的演示程序。程序由甲、乙两位程序员分工协作，共同编写
- C++ → Java



# 3.1 面向对象程序设计方法

## C/C++语法：定义函数

```
函数类型 函数名(形式参数列表)
{
    函数体
}
```

语法说明：

- **函数类型**指定函数返回值（即函数值）的数据类型。函数类型由函数功能决定，可以是除数组之外的任何数据类型，省略时默认为int型。某些函数可能只是完成某种功能，但没有返回值，此时函数类型应定义为void。
- **函数名**指定函数的名称，由程序员命名，需符合标识符的命名规则。通常函数之间不能重名。
- **形式参数列表**定义了函数接收输入参数所需的变量，这些变量称为**形式参数**，简称为**形参**。可以有多个形参，每个形参以“**数据类型 变量名**”的形式定义，形参之间用“,”隔开。某些函数可能不需要输入参数，此时形式参数列表省略为空。
- **函数体**是描述数据处理算法的C/C++语句序列，用大括号“{ }”括起来。函数体中可以定义专供本函数使用的局部变量。如果函数有返回值，则应使用return语句返回，返回值的数据类型应与函数类型一致。
- “**函数类型 函数名(形式参数列表)**”是函数的头部，被称为**函数原型**（prototype）或**函数签名**（signature）。它定义了函数的调用接口，即函数名、输入参数和返回值类型。



# 3.1 面向对象程序设计方法

## • 结构化程序设计中的函数

C/C++语法：调用函数

函数名( 实际参数列表 )

语法说明：

- 函数名指定被调用函数的名称。
- 实际参数列表给出函数所需要的输入参数。调用函数时应按被调用函数的要求给定的输入参数值，这些参数值称为**实际参数**，简称为**实参**。实际参数可以是常量、变量或表达式，参数之间用“,”隔开。调用时，计算机首先将实参值按位置顺序一一赋值给对应的形参变量，这称为函数调用时的**参数传递**。实参与形参应当个数一致，类型一致。
- “函数名( 实际参数列表 )”就是在调用某个函数。有返回值的函数调用可作为操作数参与表达式运算，该操作数等于函数定义里的返回值。某些函数可能只是完成某种功能，但没有返回值。无返回值的函数调用直接加分号“;”即构成一条完整的函数调用语句。
- 一个函数调用另一个函数，调用别人的函数称为主调函数，被调用的函数称为被调函数。
- 调用函数前，需要编写声明语句对被调函数进行声明。
- 函数声明语句就是函数原型加上分号，即“函数类型 函数名(形式参数列表) ;”。将多条函数声明语句集中放在某个头文件 (.h) 中，然后使用“#include”指令插入头文件，这样可以简化函数声明。



中國農業大學

閻道宏

# 3.1 面向对象程序设计方法

- 结构化程序设计中的函数

例3-1 计算长方形面积和周长的C++程序代码（函数）

程序员甲：主函数（1.cpp）

```
1  #include <iostream> // C++语言的头文件
2  using namespace std; // 声明命名空间
3  // C语言: #include <stdio.h>
4  #include "2.h"      // 插入头文件2.h
5
6  int main( )
7  {
8      int a, b; // 定义保存长宽数据的变量
9      cin >> a >> b; // 输入长方形的长宽
10     // C语言: scanf( "%d %d", &a, &b );
11
```

程序员乙：子函数（2.cpp）

```
// 计算长方形面积和周长的函数
int Area(int length, int width)
{
    return ( length*width );
}
int Len(int length, int width)
{
    return ( 2*(length+width) );
}
```

程序员乙：头文件（2.h）

在结构化程序设计方法中，**函数**是重用算法代码的基本语法形式。

两类不同的“**程序员角色**”

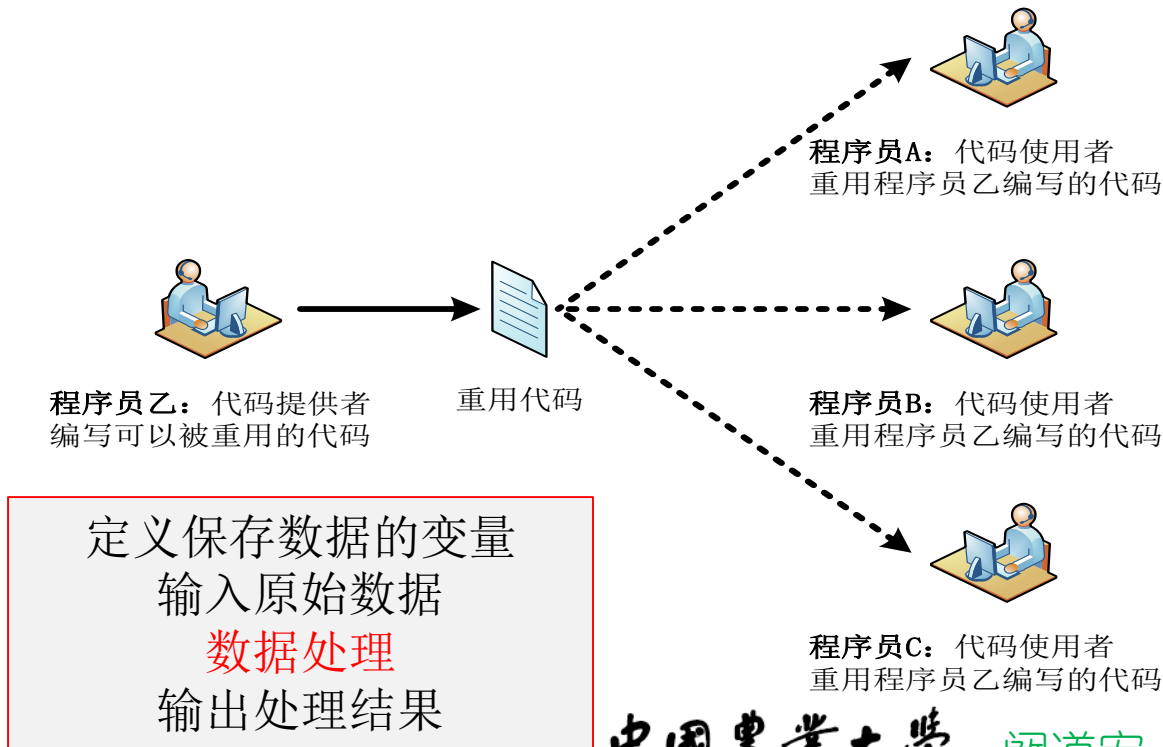


中國農業大學

閻道宏

# 3.1 面向对象程序设计方法

- 结构化程序设计中的函数
  - 两类程序员角色：代码提供者、代码使用者





# 3.1 面向对象程序设计方法

例3-2 计算长方形面积和周长的C++程序代码（结构体类型）

程序员甲：主函数（1.cpp）

```
1  #include <iostream>
2  using namespace std;
3  #include "2.h" // 插入头文件2.h
4
5  int main( )
6  {
7      // int a, b; // 删除该定义变量语句
8      // 改用结构体类型Rectangle定义变量
9      struct Rectangle rect; // 定义结构体变量
10
11     cin >> rect.a >> rect.b;
12     // 用rect的下属成员a保存长度
13     // 用rect的下属成员b保存宽度
```

程序员乙：子函数（2.cpp）

```
// 计算长方形面积和周长的函数
int Area(int length, int width)
{ return ( length*width ); }
int Len(int length, int width)
{ return ( 2*(length+width) ); }
```

程序员乙：头文件（2.h）

```
// 定义一个长方形结构体类型
struct Rectangle
{
    int a; // 保存长度的成员a
    int b; // 保存宽度的成员b
```

结构体类型是结构化程序设计中重用数据代码

“分类管理”

定义保存数据的变量  
输入原始数据  
数据处理  
输出处理结果



# 3.1 面向对象程序设计方法

例3-3 计算长方形面积和周长的C++程序代码（类与对象）

程序员甲：主函数（1.cpp）

```
1  #include <iostream>
2  using namespace std;
3  #include "2.h" // 插入头文件2.h
4
5  int main( )
6  {
7      // struct Rectangle rect; // 删除该语句
8      // 改用类Rectangle定义变量（即对象）
9      Rectangle rect; // 定义一个长方形对象rect
10
11     cin >> rect.a >> rect.b;
12     // 用rect的数据成员a保存长度
13     // 用rect的数据成员b保存宽度
14
15     // 调用rect的函数成员求其面积和周长
16     cout << rect.Area( ) << endl; // 不需要传递长宽
17     cout << rect.Len( ) << endl;
18     return 0;
19 }
20
```

程序员乙：类实现程序文件（2.cpp）

```
#include "2.h" // 插入头文件2.h
// 定义长方形类Rectangle：类实现部分
// 给出各函数成员的完整定义代码
int Rectangle::Area( ) // 不需要传递长宽
{ return ( a*b ); }
int Rectangle::Len( )
{ return ( 2*(a+b) ); }
```

程序员乙：类声明头文件（2.h）

```
// 定义一个长方形类Rectangle
// 定义类的代码分为声明和实现两部分
class Rectangle // 类声明部分
{
public:
    int a; // 数据成员：保存长度
    int b; // 数据成员：保存宽度
    int Area( ); // 函数成员：计算面积
    int Len( ); // 函数成员：计算周长
};
// 类Rectangle的实现部分放在2.cpp文件中
```



中国农业大学

阚道宏

# 3.1 面向对象程序设计方法

- 结构化程序设计方法
  - 函数是分解出的算法零件
  - 结构体类型是分解出的数据零件
  - 程序升级

```
struct Rectangle // 修改成员a、b的数据类型
{
    double a; // 原来为: int a;
    double b; // 原来为: int b;
};
```

```
double Area(double length, double width) // 原来为: int Area(int length, int width)
{ return ( length*width ); }
double Len(double length, double width) // 原来为: int Len(int length, int width)
{ return ( 2*(length+width) ); }
```

```
double Area(double length, double width); // 原来为: int Area(int length, int width);
double Len(double length, double width); // 原来为: int Len(int length, int width);
```



中國農業大學

閻道宏

# 3.1 面向对象程序设计方法

- 面向对象程序设计方法

- 数据类=数据+算法

- 程序升级

class Rectangle // 修改头文件2.h中的类声明部分

```
{  
public:  
    double a; // 原来为: int a;  
    double b; // 原来为: int b;  
    double Area(); // 原来为: int Area();  
    double Len(); // 原来为: int Len();  
};
```

// 修改程序文件2.cpp中的类实现部分

```
double Rectangle::Area() // 原来为: int Rectangle::Area()  
{ return ( a*b ); }  
double Rectangle::Len() // 原来为: int Rectangle::Len()  
{ return ( 2*(a+b) ); }
```

- 参数传递少
- 代码集中
- 牵涉程序员少



中國農業大學

閻道宏

# 3.1 面向对象程序设计方法

例3-4 计算长方形面积和周长的C++程序代码（添加输入输出功能）

程序员甲：主函数（1.cpp）

```
1 // #include <iostream> // 删除这2条语句
2 // using namespace std;
3 // 主函数不再使用cin/cout, 删除上面2条语句
4
5 #include "2.h" // 插入头文件2.h
6
7 int main( )
8 {
9     // 使用功能完善后的类Rectangle定义对象
10    Rectangle rect; // 定义一个长方形对象rect
11
12    // cin>>rect.a>>rect.b; // 删除该语句
13    rect.Input(); // 调用Input成员输入长宽
14
15    // cout<<rect.Area()<<endl; // 删除该语句
16    // cout<<rect.Len()<<endl; // 删除该语句
17    rect.Output(); // 调用Output成员输出结果
18
19    return 0;
20 }
```

定义保存数据的变量  
输入原始数据  
数据处理  
输出处理结果

程序员乙：类实现程序文件（2.cpp）

```
#include <iostream>
using namespace std;
// 本程序需使用cin/cout, 添加上面2条语句
#include "2.h" // 插入头文件2.h
// 定义长方形类Rectangle: 类实现部分
double Rectangle::Area( )
{ return ( a*b ); }
double Rectangle::Len( )
{ return ( 2*(a+b) ); }
void Rectangle::Input( ) // 输入长宽
{ cin >> a >> b; }
void Rectangle::Output( ) // 输出面积和周长
{
    cout << Area( ) << endl;
    cout << Len( ) << endl;
}
```

程序员乙：类声明头文件（2.h）

```
// 为长方形类Rectangle添加2个函数成员
class Rectangle // 类声明部分
{
public:
    double a, b; // 数据成员: 保存长度和宽度
    double Area( ); // 函数成员: 计算面积
    double Len( ); // 函数成员: 计算周长
    void Input( ); // 函数成员: 输入长宽
    void Output( ); // 函数成员: 输出结果
};
```

# 3.1 面向对象程序设计方法

- 面向对象程序设计中的封装

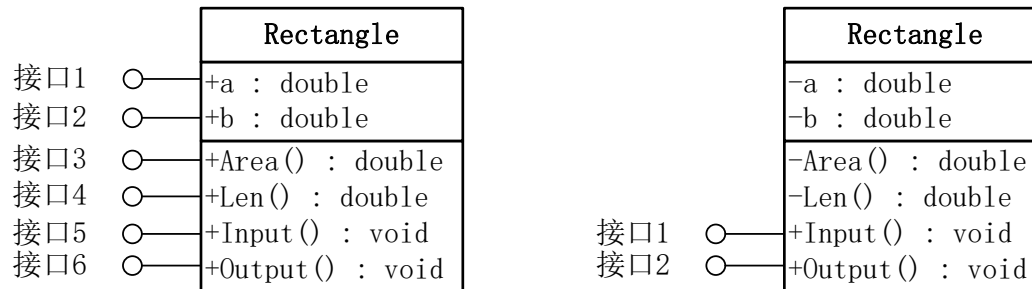
定义类的程序员可以将需要被外部直接访问的成员开放出来，同时将不需要被直接访问的成员隐藏起来，这就是面向对象程序设计中类的**封装**。

- **开放**。定义类时将必须被外部访问的成员开放出来，以保证类的功能可以被正常使用
- **隐藏**。定义类时将不需要被外部访问的成员隐藏起来，以防止它们被误访问
- 公有权限**public**。被赋予公有权限的类成员是开放的
- 私有权限**private**。被赋予私有权限的类成员将被隐藏



# 3.1 面向对象程序设计方法

- 面向对象程序设计中的封装



```
class Rectangle // 在类声明部分设定各成员的访问权限
{
    private: // 以下4个成员被设定为私有权限
        int a, b; // 数据成员：保存长度和宽度
        int Area(); // 函数成员：计算面积
        int Len(); // 函数成员：计算周长
    public: // 以下2个成员被设定为公有权限
        void Input(); // 函数成员：输入长和宽
        void Output(); // 函数成员：输出面积和周长
};
```

```
Rectangle rect;
rect.Input();
rect.Output();

cin >> rect.a >> rect.b;
```

```
void Rectangle::Input()
{
    cout << "请输入长和宽："; cin >> a >> b;
    while (a < 0 || b < 0) // 数据合法性检查
    {
        cout << "长宽值不能为负数，请重新输入：";
        cin >> a >> b; }
}
```



# 3.1 面向对象程序设计方法

例3-5 计算长方形面积和周长的Java程序代码（类与对象）

程序员甲：主类文件（RectangleTest.java）

```
1 public class RectangleTest { // 主类
2
3     // 将主函数main()定义在类中
4     public static void main(String[] args) {
5         // Java需要动态创建对象
6         Rectangle obj = new Rectangle();
7
8         obj.Input(); // 输入长宽
9         obj.Output(); // 显示结果
10    }
11
12 }
13
14
15
16
17
18
19
```

程序员乙：长方形类文件（Rectangle.java）

```
import java.util.Scanner; // 导入外部程序Scanner

public class Rectangle { // 长方形类定义代码
    private double a, b; // 字段：保存长度和宽度
    private double Area() // 方法：计算面积
    { return a*b; }
    private double Len() // 方法：计算周长
    { return 2*(a+b); }

    public void Input() { // 方法：输入长宽
        // 创建键盘扫描器对象
        Scanner sc = new Scanner( System.in );
        // 然后通过键盘扫描器对象输入长宽
        a = sc.nextDouble(); b = sc.nextDouble();
    }
    public void Output() { // 方法：输出结果
        System.out.println( Area() +", " +Len() );
    }
}
```



中國農業大學

閻道宏



# 3.1 面向对象程序设计方法

- Java语言中的类与对象

例3-6 将主方法main()定义在长方形类Rectangle中

```
1 import java.util.Scanner; // 导入外部程序Scanner
2
3 public class Rectangle { // 长方形类定义代码
4     private double a, b; // 字段：保存长度和宽度
5     private double Area() { ..... } // 代码省略
6     private double Len() { ..... } // 代码省略
7     public void Input() { ..... } // 代码省略
8     public void Output() { ..... } // 代码省略
9
10    // 将主方法main()定义在长方形Rectangle类中
11    public static void main(String[] args) {
12        Rectangle obj = new Rectangle();
13        obj.Input(); // 输入长宽
```

面向对象程序设计：分类（抽象）、封装、继承、多态



## 3.2 面向对象程序的设计过程

- 程序设计任务



- 面向对象程序的设计过程：分析、抽象、组装
- 统一建模语言UML（Unified Modeling Language）
- Java语言



## 3.2 面向对象程序的设计过程

- 分析
  - **用户**启动**测算程序**，由计算机执行这个程序。
  - 程序等待用户输入原始数据，其中包括养鱼池的长宽、清水池和污水池的半径。用户输入数据后，程序继续执行。
  - 程序要在计算机中模拟创建**养鱼池**、**清水池**和**污水池**，然后计算并汇总其造价。
  - 显示汇总后的工程总造价。测算程序结束。

客观事物？



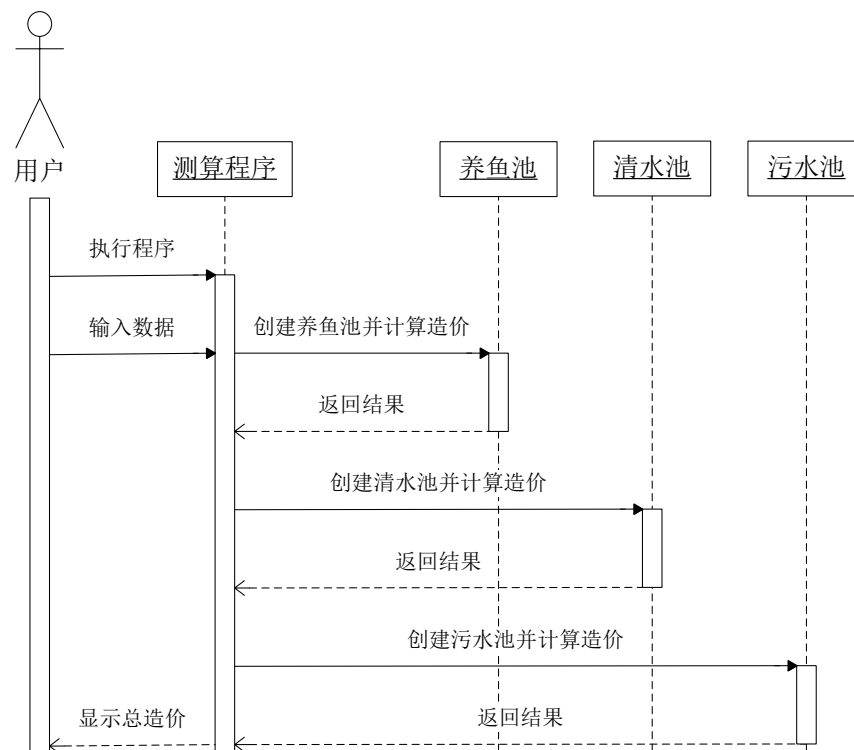
中國農業大學

閻道宏

## 3.2 面向对象程序的设计过程

- 分析
- 结构化程序设计  
– 数据和算法

```
double length, width;  
double RectCost(double a, double b)  
{  
    double cost ;  
    cost = a * b * 10 ;  
    return cost ;  
}
```



数据模型？



## 3.2 面向对象程序的设计过程

- 抽象

为客观事物建立数据模型。

- 属性（property）

- 方法（method）

- 客观对象

- 分类：具有相同数据模型的客观对象

RectPool
+a : double
+b : double
+RectCost() : double

CirclePool
+r : double
+CircleCost() : double

CirclePool
+r : double
+CircleCost() : double
+FenceCost() : double



中國農業大學

阚道宏

## 3.2 面向对象程序的设计过程

- 抽象

- Java语言支持面向对象程序设计

- 类图：类

- 属性：字段

- 方法：方法

- 封装：访问权限

RectPool
+a : double
+b : double
+RectCost() : double

CirclePool
+r : double
+CircleCost() : double

```
class RectPool {  
    public double a, b;  
    public double RectCost( )  
    { return (a*b *10); }  
}
```

```
class CirclePool {  
    public double r;  
    public double CircleCost( )  
    { return (3.14 * r*r *10); }  
}
```



## 3.2 面向对象程序的设计过程

- 组装
  - 类：描述客观对象数据模型的图纸
  - 对象：按照图纸生产出的产品，图纸的实例
  - 用类定义对象

```
CirclePool cObj1, cObj2;  
cObj1 = new CirclePool( ); // 清水池对象  
cObj2 = new CirclePool( ); // 污水池对象
```

内存对象

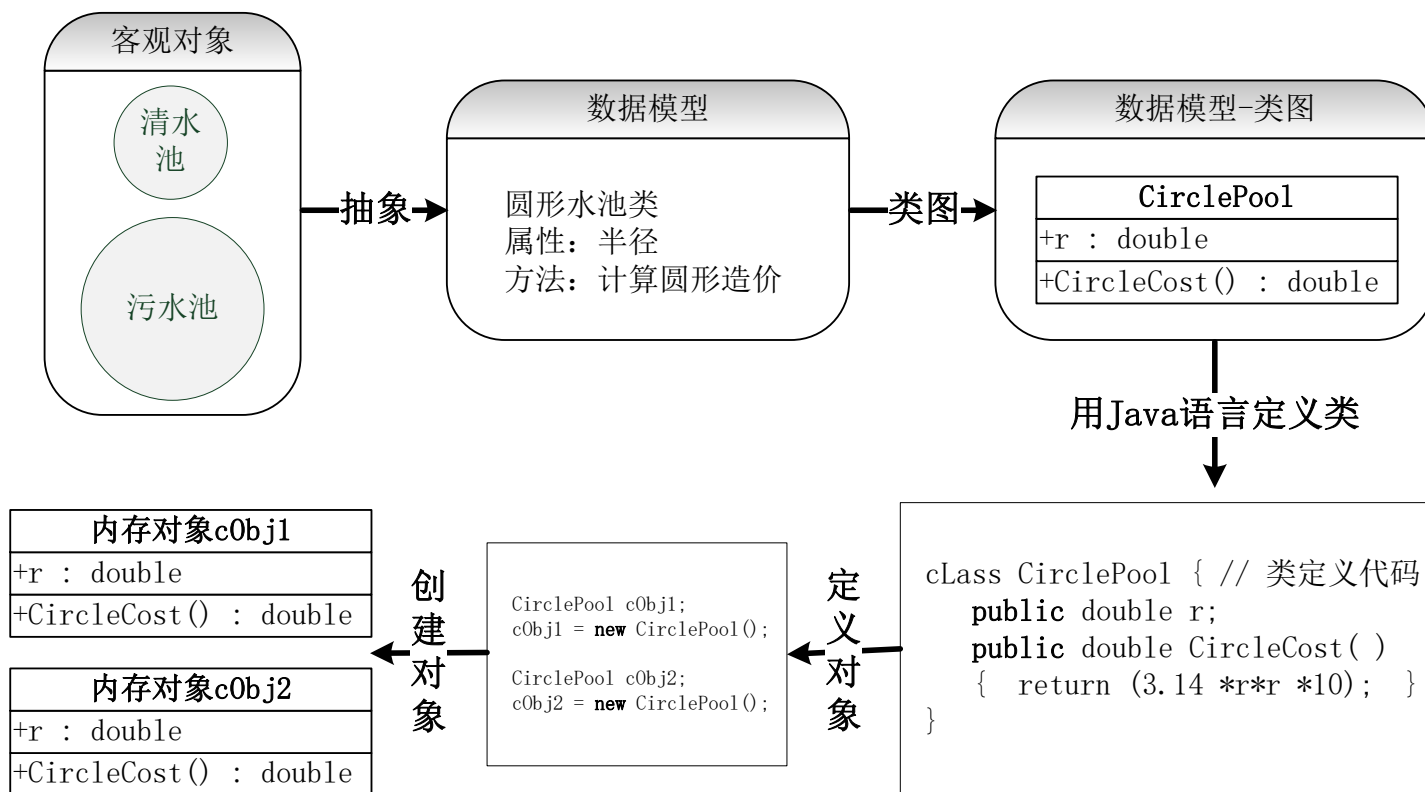


中國農業大學

阚道宏

## 3.2 面向对象程序的设计过程

- 从客观对象到内存对象





## 3.2 面向对象程序的设计过程

- 组装

- 访问对象

```
// 访问清水池对象cObj1  
cObj1.r = sc.nextDouble();  
totalCost += cObj1.CircleCost( );
```

```
// 访问污水池对象cObj2  
cObj2.r = sc.nextDouble();  
totalCost += cObj2.CircleCost( );
```

- 只能访问公有成员



### 例3-7 测算养鱼池工程总造价的Java程序代码（面向对象程序设计方法）

程序员甲：主类+主方法（Pool.Java）

```

1  import java.util.Scanner; // 导入外部程序Scanner
2
3  public class Pool {    // 主类
4      public static void main(String[] args) { // 主方法
5          Scanner sc = new Scanner( System.in );
6          double totalCost = 0; // 保存总造价的变量
7          // 处理长方形养鱼池
8          RectPool rObj;    // 定义引用
9          rObj = new RectPool(); // 创建长方形鱼池对象
10         rObj.a = sc.nextDouble(); // 输入长宽值
11         rObj.b = sc.nextDouble();
12         totalCost += rObj.RectCost(); // 汇总造价
13         // 处理清水池和污水池
14         CirclePool cObj1, cObj2; // 定义引用
15         cObj1 = new CirclePool(); // 创建清水池对象
16         cObj2 = new CirclePool(); // 创建污水池对象
17         cObj1.r = sc.nextDouble(); // 输入清水池半径
18         cObj2.r = sc.nextDouble(); // 输入污水池半径
19         totalCost += cObj1.CircleCost(); // 汇总造价
20         totalCost += cObj2.CircleCost(); // 汇总造价
21         // 显示总造价totalCost
22         System.out.println( totalCost );
23     }
24 }
```

程序员乙：长方形养鱼池类（RectPool.java）

```

public class RectPool { // 长方形养鱼池类
    public double a, b; // 字段：长宽
    public double RectCost() // 计算造价
    { return (a*b *10); }
}
```

程序员乙：圆形水池类（CirclePool.java）

```

public class CirclePool { // 圆形水池类
    public double r; // 字段：半径
    public double CircleCost() // 计算造价
    { return (3.14 * r*r *10); }
}
```



中國農業大學

閻道宏

## 3.3 类与对象的语法细则

Java语法：定义类

```
[public] class 类名 {  
    [访问权限] 数据类型 字段名 [= 初始值];  
    .....  
    [访问权限] 返回值类型 方法名( 形式参数列表 ){  
        方法体  
    }  
    .....  
}
```

语法说明：

- 定义类时使用关键字**class**。通常在**class**之前使用关键字**public**将类的访问权限设定为公有，也可以省略（此时类将具有默认访问权限）。注：Java语法的中括号“[ ]”表示其中的内容可省略，下同。
- 类名需符合标识符的命名规则，习惯上以大写字母开头。
- 类的下属成员有两种，分别是**字段**（存储数据）和**方法**（处理数据的算法）。某些特殊的类可能只包含一种成员，比如只包含字段，或只包含方法。
- 类成员的**访问权限**有4种，分别是公有权限（**public**）、保护权限（**protected**）、私有权限（**private**）或默认权限（未指定访问权限）。
- 方法成员可以访问本类中任意位置的字段（字段相当于是类中的全局变量），或调用本类中任意位置的其他方法。类成员之间互相访问不需要“先定义，后访问”，也不受权限约束。



## 3.3 类与对象的语法细则

例3-8 一个钟表类Clock的Java示意代码（Clock.java）

```
1 import java.util.Scanner; // 导入外部程序Scanner
2
3 public class Clock {    // 定义钟表类Clock
4     // 将字段设为private权限，即私有成员
5     private int hour;    // 字段hour：保存小时数
6     private int minute; // 字段minute：保存分钟数
7     private int second; // 字段second：保存秒数
8     // 将方法设为public权限，即公有成员
9     public void set() { // 不带参数的方法set()：从键盘输入时间
10         Scanner sc = new Scanner( System.in ); // 创建键盘扫描器对象sc
11         hour = sc.nextInt(); minute = sc.nextInt(); second = sc.nextInt();
12     }
13     public void set(int h, int m, int s) { // 带参数的方法set()：用参数设定时间
14         hour = h; minute = m; second = s; // 将参数分别赋值给对应的数据成员
15     }
16     public void show() { // 方法show：显示时间
17         System.out.println( hour + " : " + minute + " : " + second ); // 显示格式：时:分:秒
18     }
19 }
```



## 3.3 类与对象的语法细则

- 类的定义
  - 字段成员的语法细则
    - 字段初始化

```
private int hour = 0; // 将钟表的时间初始化为0:0:0
private int minute = 0;
private int second = 0;
```

- 空值

表3-1 不同数据类型的空值

	byte/short/int/long	float/double	char	boolean	引用变量
字段默认值	0	0.0	'\u0000'	false	null

- 只读字段final

```
private final int second = 0; // 将字段second定义成只读字段
```



## 3.3 类与对象的语法细则

- 类的定义

- 方法成员的语法细则

- 方法的4大要素

- 方法名
      - 形式参数列表
      - 方法体
      - 返回值类型

- 方法签名: **返回值类型** 方法名(**形式参数列表**)

- 重载方法

```
public void set( ) { // 不带参数的方法set(): 从键盘输入时间
    Scanner sc = new Scanner( System.in );
    hour = sc.nextInt(); minute = sc.nextInt(); second = sc.nextInt();
}
public void set(int h, int m, int s) { // 带参数的方法set(): 用参数设定时间
    hour = h; minute = m; second = s;
}
```



## 3.3 类与对象的语法细则

- 类的定义
  - 类成员访问权限的语法细则
    - 公有权限public
    - 私有权限private
    - 保护权限protected
    - 默认权限（未指定访问权限）



## 3.3 类与对象的语法细则

- 对象的定义与访问

- 定义对象

- 运算符**new**

- 引用及引用变量

`Clock obj1; // 预先定义一个Clock类型的引用变量obj1`  
`// 此时obj1的引用值为null，即还未引用任何对象`  
`obj1 = new Clock( );`

上述两条语句可简写为一条语句：

`Clock obj1 = new Clock( );`

- 请注意：“=”两边的类型应当一致





## 3.3 类与对象的语法细则

- 对象的定义与访问
  - 访问对象
    - 成员运算符 “.”
      - 对象名.字段名
      - 对象名.方法名(实参列表)



## 3.3 类与对象的语法细则

- 访问钟表对象的公有成员

```
Clock obj1 = new Clock( ); // 创建一个钟表对象obj1
obj1.set( );              // 调用对象obj1的公有方法set(), 输入时分秒数据
obj1.show( );             // 调用对象obj1的公有方法show(), 显示其时间
```

- 可以用钟表类Clock定义（生产）多个钟表对象

```
Clock obj2 = new Clock( ); // 创建第二个钟表对象obj2
obj2.set( 8, 30, 15 );     // 调用对象obj2的公有方法set(), 设置时间8:30:15
obj2.show( );              // 调用对象obj2的公有方法show(), 显示其时间
```

- 一个对象可以被多次引用

```
Clock obj; // 再定义一个Clock类的引用变量obj
obj = obj1; // 赋值后, obj与obj1引用同一个对象, 该对象被引用了2次
obj.set( 12, 0, 0 ); // 通过引用变量obj操作钟表对象, 将其时间设为12:0:0
obj.show( ); // 显示对象的时间, 显示结果应为: 12:0:0
obj1.show( ); // 显示obj1所引用对象的时间, 显示结果也为: 12:0:0
```



## 3.3 类与对象的语法细则

- 引用数据类型
  - **基本**数据类型（primitive data type）  
byte、short、int、long、float、double、char、boolean
  - Java语言中，使用基本数据类型定义**变量**，定义时直接为变量分配内存单元
  - 后续程序将通过**变量名**访问变量的内存单元



## 3.3 类与对象的语法细则

- 引用数据类型（reference data type）  
类类型、数组类型、接口类型、枚举类型等
  - 先定义引用数据类型的引用变量
  - 再用运算符new创建引用数据类型的对象，并将所返回的对象引用保存到引用变量中
  - 后续程序通过引用变量访问对象及其下属成员



# 3.3 类与对象的语法细则

- 引用数据类型

- 变量及对象的内存

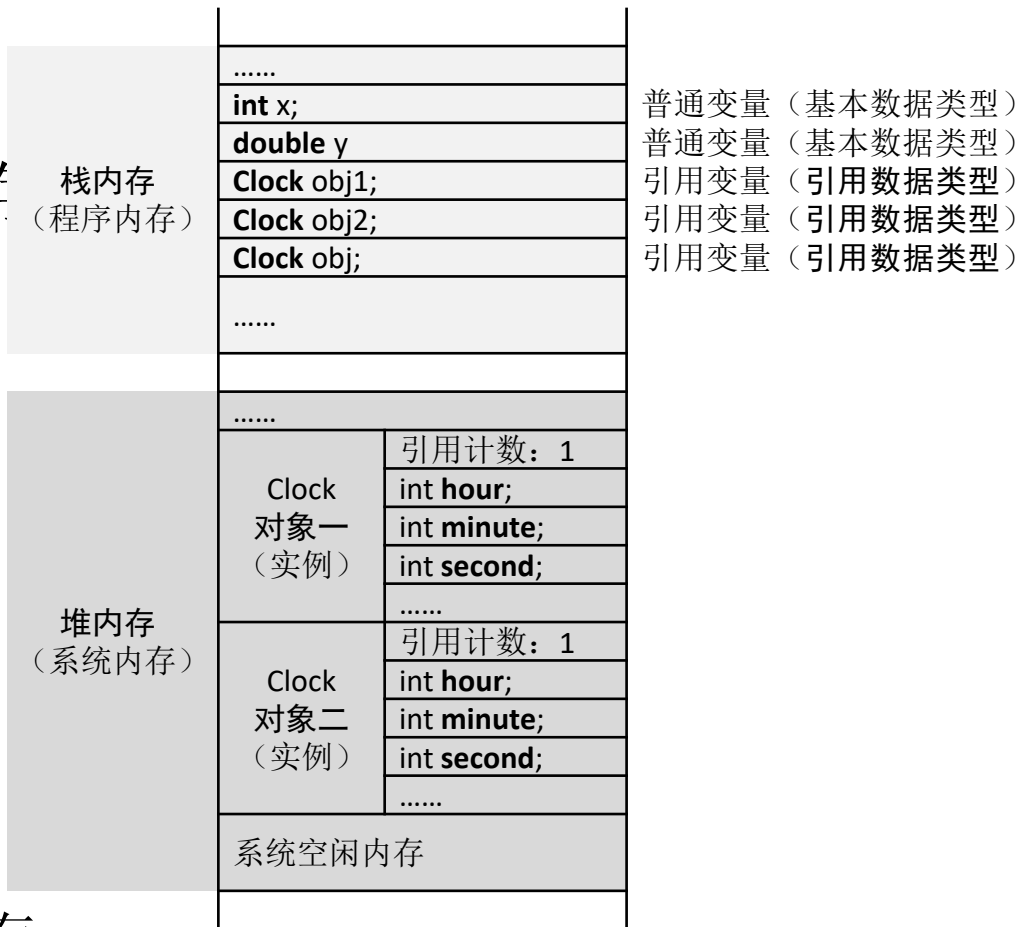
- 变量

**int x;**

**double y;**

**Clock obj1;**

- 局部变量：**栈**内存



# 3.3 类与对象的语法细则

- 引用数据类型
  - 变量及对象的内存分配

## — 对象

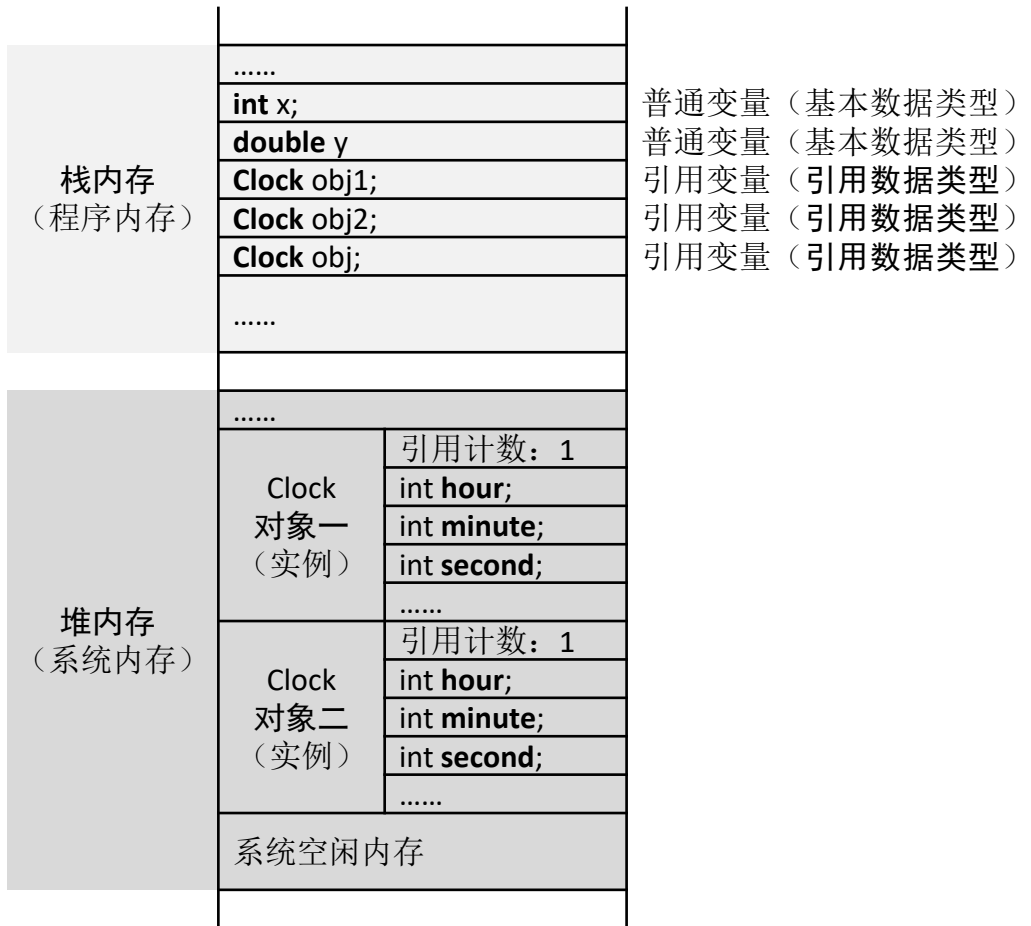
```
obj1 = new Clock( );  
Clock obj2 = new Clock( );
```

```
Clock obj;  
obj = obj1;
```

```
obj = obj2;
```

```
obj = null;
```

```
obj2 = null;
```



- Java语言中的垃圾回收 (garbage collection) 机制



## 3.3 类与对象的语法细则

- 三种不同的变量
  - **字段**：定义在类中的变量成员，用于保存属性数据。字段相当于类中的全局变量，可以被类中的所有方法成员访问
  - **局部变量**：类中方法成员在方法体中定义的变量，仅能在所定义的方法体或复合语句中访问
  - **形式参数**（形参）：类中方法成员在头部小括号里面定义的变量，用于接收原始数据。形参仅能在所定义的方法体中访问

**注：**Java语言中没有全局变量的概念



中國農業大學

閻道宏

## 3.3 类与对象的语法细则

- 三种不同的变量

例3-10 一个为钟表设置整点时间的Java示例代码（ClockTest.java）

```
1 import java.util.Scanner; // 导入外部程序Scanner
2
3 public class ClockTest { // 主类
4     public static void main(String[] args) { // 主方法
5         int hour; // 局部变量：普通变量，未初始化
6         Clock c1; // 局部变量：引用变量，未初始化
7         hour = 12; c1 = new Clock( ); // 为局部变量赋值
8         c1.set( 8, 30, 15 ); // 设置钟表对象c1的时间
9         c1.show( ); // 显示结果：8:30:15
10
11         Clock c2; // 再定义一个局部引用变量c2
12         c2 = setHour( c1, hour ); // 调用setHour()将c1设为整点，并将返回值赋值给c2
13         c2.show( ); // 显示c2的时间，结果应为：12:0:0
14         c1.show( ); // 显示c1的时间，结果也为：12:0:0
15     }
16
17     private static Clock setHour( Clock rc, int h ) { // 将钟表时间设为整点的方法
18         rc.set( h, 0, 0 ); // 设置rc的时间，小时数为接收的参数h，分钟和秒数设为0
19         return rc;
20     } }
```

• 传递**基本数据类型**数据时直接传递数值，即**值传递**。

• 传递**引用数据类型**的对象时所传递的是对象引用（不是对象本身），这被称为是**引用传递**。





## 3.3 类与对象的语法细则

- 类与对象的编译原理
  - 类代码的编译

例3-11 钟表类Clock中方法成员的编译举例

方法成员set(int h, int m, int s): 调整前		方法成员set(int h, int m, int s): 调整后	
1	public void <b>set</b> ( int h, int m, int s ) {		public void <b>set</b> ( Clock <b>this</b> , int h, int m, int s ) {
2	hour = h;		<b>this</b> .hour = h;
3	minute = m;		<b>this</b> .minute = m;
4	second = s;		<b>this</b> .second = s;
5	}		}
方法成员show( ): 调整前		方法成员Show( ): 调整后	
1	public void <b>show</b> ( ) {		public void <b>show</b> ( Clock <b>this</b> )
2	System.out.println(		System.out.println(
3	hour + ":" + minute + ":" + second );		<b>this</b> .hour + ":" + <b>this</b> .minute + ":" + <b>this</b> .second );
4	}		}



## 3.3 类与对象的语法细则

- 类与对象的编译原理

- 调用对象方法成员语句的编译

```
Clock obj1 = new Clock( ); // 创建钟表对象obj1
```

```
obj1.set( 8, 0, 0 ); // 将obj1的时间设为8:0:0
```

```
obj1.show( ); // 显示obj1的时间，显示结果： 8:0:0
```

```
set( obj1, 8, 0, 0 );
```

```
void set(Clock this, int h, int m, int s) { ... }
```

```
this.hour = h; this.minute = m; this.second = s;
```

```
show( obj1 );
```



## 3.3 类与对象的语法细则

- 类与对象的编译原理
  - 同类的多个对象在内存中**共用**一份方法代码

Clock **obj1** = new Clock( ); // 创建钟表对象obj1

Clock **obj2** = new Clock( ); // 创建钟表对象obj2

**obj1.set**( 8, 0, 0 );  
**obj2.set**( 9, 30, 15 );

**set**( **obj1**, 8, 0, 0 );  
**set**( **obj2**, 9, 30, 15 );

Clock <b>obj1</b> ;		引用对象一
Clock <b>obj2</b> ;		引用对象二
对象一 (实例)	int <b>hour</b> ;	对象一的内存单元
	int <b>minute</b> ;	
	int <b>second</b> ;	
对象二 (实例)	int <b>hour</b> ;	对象二的内存单元
	int <b>minute</b> ;	
	int <b>second</b> ;	

void **set**(Clock **this**, int h, int m, int s) { ... }



## 3.3 类与对象的语法细则

- 类与对象的编译原理

- 程序员在编写方法成员代码时，形参**this**是隐含的，在方法体中访问其他类成员也不需要添加**this**引用，这些都由编译器在编译时自动添加
- 程序员可以在访问类成员时显式添加“**this.**”，也可以通过**this**获取当前对象的引用，或把**this**作为实参将当前对象的引用继续传递给其他方法
- 程序员还可以利用**this**来区分与形参或局部变量重名的字段

```
public void set( int hour, int minute, int s ) {  
    this.hour = hour;           // “this.hour” 指代的是字段hour  
    this.minute = minute;       // “this.minute” 指代的是字段minute  
    second = s;                 // 形参s与字段second不重名，可以不用this  
}
```



## 3.3 类与对象的语法细则

- 类的构造方法
  - 构造方法（**constructor**）是类中一种特殊的方法
    - 构造方法的名字必须与类名相同
    - 构造方法通过形参传递初始值，实现对新建对象字段成员的初始化
    - 构造方法可以重载，即定义多个同名的构造方法，这样可以提供多种形式的初始化方法
    - 构造方法没有返回值，定义时不能写返回值类型，写 **void** 也不行
    - 构造方法通常是类外调用，其访问权限不能设为 **private**

类名() {} // 默认构造方法



中國農業大學

阚道宏

## 3.3 类与对象的语法细则

- 类的构造方法
  - 初始化对象

```
public Clock( int p1, int p2, int p3 ) { // 带形参的构造方法
    hour = p1; minute = p2; second = p3;
}
```

```
public Clock( ) { // 不带形参的构造方法
    this( 0, 0, 0 ); // 调用本类重载的带形参构造方法（须为第一条语句）
    // hour = 0; minute = 0; second = 0; // 或直接赋值
}
```

```
Clock obj1 = new Clock( 8, 30, 15 ); // 给了实参，将调用带形参的构造方法
Clock obj2 = new Clock( );          // 未给实参，将调用不带形参的构造方法
```

```
Clock obj = obj1;                    // 未实际创建对象，不会调用构造方法
```



## 3.3 类与对象的语法细则

- 类的构造方法
  - 拷贝构造方法

```
public Clock( Clock oldObj ) {  
    hour    = oldObj.hour;  
    minute  = oldObj.minute;  
    second  = oldObj.second;  
}
```

```
Clock obj1 = new Clock( 8, 30, 15 );  
Clock obj2 = new Clock( obj1 );
```



## 3.3 类与对象的语法细则

- 类的构造方法

- 显示对象的创建过程

```
public Clock() { // 不带形参的构造方法
    hour = 0; minute = 0; second = 0;
    System.out.println( "Clock( ) called." ); // 添加显示信息的语句
}
public Clock( int p1, int p2, int p3 ) { // 带形参的构造方法
    hour = p1; minute = p2; second = p3;
    System.out.println( "Clock(int p1, int p2, int p3) called." ); // 添加显示信息的语句
}
public Clock( Clock oldObj ) { // 拷贝构造方法
    hour = oldObj.hour; minute = oldObj.minute; second = oldObj.second;
    System.out.println( "Clock( Clock oldObj ) called." ); // 添加显示信息的语句
}
```

Clock **obj1** = **new** Clock( ); // 未给实参: Clock( ) called.

Clock **obj2** = **new** Clock( 8, 30, 15 ); // 给3个int型实参: Clock(int p1, int p2, int p3) called.

Clock **obj3** = **new** Clock( obj2 ); // 给1个Clock型实参: Clock( Clock oldObj ) called.



中國農業大學

閻道宏



## 3.3 类与对象的语法细则

- 类的静态成员
  - 全局变量、外部函数
  - 纳入到类中管理：静态成员

例3-12 对钟表类Clock进行对象计数的伪代码（模拟C/C++语言）

```
1  int totalClock = 0; // 模拟C++语言：定义一个全局变量，记录所创建的Clock对象个数
2  void plusObj( ) { totalClock ++; } // 模拟C/C++语言：定义一个外部函数，将计数加1
3
4  public class Clock { // 钟表类Clock
5      private int hour, minute, second;           // 字段成员
6      public void set( ) { ..... }                // 不带参数的设置时间方法set（代码省略）
7      public void set(int h, int m, int s) { ..... } // 带参数的设置时间方法set（代码省略）
8      public void show( ) { ..... }                // 显示时间方法show（代码省略）
9      public Clock( ) // 定义一个构造方法
10     { .....; plusObj( ); } // 希望通过构造方法为钟表对象增加计数功能
11 }
```



## 3.3 类与对象的语法细则

- 类的静态成员
  - Java语言不允许在类外定义全局变量或函数

例3-13 通过静态成员实现对钟表类Clock进行对象计数的Java示意代码（Clock.java）

```
1 public class Clock { // 定义钟表类Clock
2     public static int totalClock = 0; // 定义一个静态字段，记录已创建的Clock对象个数
3     private static void plusObj( ) { totalClock++; } // 定义一个静态方法，将计数加1
4
5     private int hour, minute, second;    // 字段成员
6     public void set( ) { ..... }        // 不带参数的设置时间方法set（代码省略）
```

```
public class ClockTest { // 测试类
    public static void main(String[] args) { // 主方法
        Clock c1 = new Clock( ); // 创建第一个对象c1
        Clock c2 = new Clock( ); // 创建第二个对象c2
        System.out.println( Clock . totalClock ); // 通过类名访问静态成员totalClock
        System.out.println( c1 . totalClock ); // 或通过任一对象引用（c1或c2）访问静态成员
        System.out.println( c2 . totalClock );
    } }
```



## 3.3 类与对象的语法细则

- 类的静态成员
  - 静态字段的内存分配

Clock **c1** = new Clock( );

Clock **c2** = new Clock( );

Clock <b>c1</b> ;		引用对象一
Clock <b>c2</b> ;		引用对象二
static int totalClock;		静态字段（类字段）
对象一 （实例）	int hour;	普通字段（实例字段）
	int minute;	
	int second;	
对象二 （实例）	int hour;	普通字段（实例字段）
	int minute;	
	int second;	

**c1**. totalClock、 **c2**. totalClock

**Clock** . totalClock



中國農業大學

阚道宏

## 3.3 类与对象的语法细则

- 类的静态成员

- 静态字段的语法细则

- **关键字static**: 在类中定义静态字段需使用关键字**static**进行限定，通常放在访问权限之后，数据类型之前。定义静态字段时可以初始化
    - **在本类访问静态字段**: 在本类的方法成员中访问静态字段，直接使用字段名访问，访问时不受权限约束。这一点与访问普通字段是一样的
    - **不能通过this访问静态字段**: 静态字段是类字段，没有包含在对象中，因此不能使用关键字**this**访问静态字段。这一点与访问普通字段是不一样的。注: **this**是指向当前对象的引用变量
    - **在类外访问静态字段**: 在类外其他方法（例如主方法）中访问静态字段需以“**类名.静态字段名**”的形式访问，或通过任何一个该类对象引用以“**对象引用名.静态字段名**”的形式访问。类外访问静态字段受权限约束。
    - 在类外访问静态字段时可以不创建对象，直接通过类名访问

**Clock. totalClock**



中國農業大學

閻道宏

# 3.3 类与对象的语法细则

- 类的静态成员

- 静态方法的语法细则

- **关键字static:** 在类中定义静态方法需使用关键字static进行限定，通常放在访问权限之后，返回值类型之前
    - **在本类调用静态方法:** 本类中的所有方法成员都可以调用静态方法。调用时直接使用方法名，并且不受访问权限约束。这一点与调用普通方法是一样的
    - **在类外调用静态方法:** 与静态字段一样，静态方法是一种类方法，与对象实例无关。对应地，非静态的普通方法只能在创建对象实例之后才能调用，因此被称为实例方法。在类外其他方法（例如主方法）中调用静态方法，需要以“**类名.静态方法名()**”的形式调用，或通过任何一个该类对象引用以“**对象引用名.静态方法名()**”的形式调用。类外调用受访问权限约束。在类外调用静态方法时可不创建对象，直接通过类名调用  
**Clock.plusObj()**
    - **静态方法访问本类其他成员:** 静态方法只能访问本类中的静态字段，不能访问实例字段，因为静态方法可以在没有创建任何对象的情况下直接调用，而实例字段必须在对象创建之后才会分配内存空间，因而不能访问。静态方法只能调用本类中的其他静态方法，不能调用实例方法，因为实例方法可能会间接访问实例字段



## 3.3 类与对象的语法细则

- 类的静态成员
  - 静态成员的应用
    - 以类的形式管理全局变量或外部函数
    - 将具有相同属性值的字段提炼出来，定义成静态字段，这样可以让所有对象共用同一个静态字段，从而减少内存占用。
    - 将专门处理静态字段的方法定义成静态方法

例3-14 一个使用数学类Math中静态成员的演示程序

```
1 public class MathTest { // 测试类：测试Java语言中数学类Math的静态成员
2
3     public static void main(String[] args) { // 主方法
4         System.out.println( "random()= " +Math.random() ); // 随机数函数（静态方法）
5         System.out.println( "random()= " +Math.random() );
6
7         System.out.println( "sqrt(36)= " +Math.sqrt(36) ); // 求平方根函数（静态方法）
8         System.out.println( "sin(30)= " +Math.sin(30 *Math.PI/180) ); // 正弦函数（静态方法）
9     }
10 }
```



## 3.4 数组

- 数组（array）
  - 元素（element）、下标（subscript）
  - 一维数组、二维数组
  - 遍历
- 定义数组
  - 先定义数组类型的引用变量

```
int iArray[]; // 定义一个int型数组的引用变量iArray
int []iArray; // 定义一个int型数组引用变量iArray
```
  - 再创建数组

```
iArray = new int[5];

int iArray[] = new int[5]; // 定义引用变量的同时创建数组
```



## 3.4 数组

- 定义数组时的初始化

```
int iArray[ ] = { 2, 4, 6 };
```

- 数组的语法细则

```
int iArray[ ] = new int[5];
```

```
int aRef[ ] = iArray;
```

```
iArray.length
```

```
aRef.length
```

int <b>iArray</b> [ ];		引用数组对象一
int <b>aRef</b> [ ];		引用数组对象一
数组 对象一	引用计数: 2	
	<b>length</b> : 5	public final int length;
	数组元素0: 0	iArray[0] 或 aRef[0]
	数组元素1: 0	iArray[1] 或 aRef[1]
	数组元素2: 0	iArray[2] 或 aRef[2]
	数组元素3: 0	iArray[3] 或 aRef[3]
	数组元素4: 0	iArray[4] 或 aRef[4]





## 3.4 数组

- 访问数组
  - 访问数组中的元素

```
int iArray[ ] = new int[5];  
iArray[0]、iArray[1]、iArray[2]、iArray[3]、iArray[4]
```

```
iArray[0] = 10 ;  
iArray[1] = 20 ;  
System.out.println( iArray[0] ); // 显示结果: 10  
System.out.println( iArray[1] ); // 显示结果: 20  
System.out.println( iArray[2] ); // 显示结果: 0
```

```
System.out.println( iArray[ 5 ] );  
System.out.println( iArray[ -1 ] );
```

```
int aRef[ ] = iArray;  
System.out.println( aRef[0] ); // 显示结果: 10
```



## 3.4 数组

- 访问数组
  - 数组的遍历

例3-15 一个遍历数组的Java演示程序（ArrayDemo.java）

```
1 public class ArrayDemo {           // 主类
2     public static void main(String[] args) { // 主方法
3         int iArray[] = { 2, 4, 6, 8, 10 };    // 定义一个int型数组iArray，定义时初始化
4         for (int n = 0; n < iArray.length; n++) // 遍历数组，逐个显示各数组元素的值
5             System.out.println( iArray[n] );    // 显示第n个元素的值
6         // 遍历数组，计算各数组元素的累加和
7         int sum = 0;
8         for (int n = 0; n < iArray.length; n++)
9             sum += iArray[n]; // 累加第n个数组元素
10        System.out.println( sum );
11
12        char cArray[] = { 'C', 'h', 'i', 'n', 'a' };    // 定义一个字符型数组cArray，定义时初始化
13        for (int n = 0; n < cArray.length; n++) {    // 遍历数组，将所有小写字母改为大写
14            if (cArray[n] >= 'a' && cArray[n] <= 'z') // 检查第n个元素是否小写字母
15                cArray[n] -= 32;                    // 如果是，则将小写字母改为大写
16        }
17        System.out.println( cArray );                // 只有字符数组才能整体输出，显示结果：CHINA
18    } }
```



中國農業大學

閻道宏

## 3.4 数组

- 访问数组
  - 增强for语句

例3-16 一个遍历数组（增强for语句）的Java演示程序（EnhancedFor.java）

```
1 public class EnhancedFor {           // 主类
2     public static void main(String[] args) { // 主方法
3         int iArray[] = { 2, 4, 6, 8, 10 }; // 定义一个int型数组iArray，定义时初始化
4         for (int x : iArray) // 增强for语句：依次将数组iArray中的元素取出来，赋值给x
5             System.out.println(x); // 显示所取出的值
6         // 遍历数组，计算各数组元素的累加和
7         int sum = 0;
8         for (int x : iArray) // 增强for语句：计算各数组元素的累加和
9             sum += x; // 累加所取出的值
10        System.out.println( sum );
11
12        char cArray[] = { 'C', 'h', 'i', 'n', 'a' }; // 定义一个字符型数组cArray，定义时初始化
13        for (char x : cArray) { // 增强for语句：依次将数组cArray中的元素取出来，赋值给x
14            if (x >= 'a' && x <= 'z') // 检查所取出的字符是否小写字母
15                x -= 32; // 将小写字母改为大写。注：此处只能修改x，无法修改数组元素
16        }
17        System.out.println( cArray ); // 显示结果：China
18        // 可以看出，当需要修改数组元素时，还是只能用普通for语句
19    } }
```



## 3.4 数组

- 可变长形参

- 通常，一个方法中的形参个数是确定的

```
int max(int x, int y) { ..... }
```

```
int max(int x, int y, int z) { ..... }
```

- 是否可以定义一个求任意多个数最大值的方法呢？

```
int max(int x1, int x2, ...) { ..... }
```

- Java可变长参数语法

```
int max(int ...varArgs) { // 在可变长形参名之前加 “...” （3个点）  
    ..... // 求最大值算法  
}
```



## 3.4 数组

### • 可变长形参

例3-17 一个具有可变长形参的求最大值方法Java演示程序（VarArgument.java）

```
1 public class VarArgument { // 主类
2     public static int max(int ...varArgs) { // 具有可变长形参的求最大值方法
3         // 可变长形参varArgs所接收到的实参是以数组形式存放的，varArgs是一个数组
4         if (varArgs.length < 1) return 0; // 如果没有传递实参，则直接返回0
5         int result = varArgs[0]; // 先假设第0个元素就是最大值
6         for (int n = 1; n < varArgs.length; n++) { // 求数组元素中的最大值
7             if (varArgs[n] > result) result = varArgs[n];
8         }
9         /* 也可使用以下的增强for语句来求最大值
10        for (int e : varArgs)
11            { if (e > result) result = e; }
12        */
13        return result;
14    }
15
16    public static void main(String[] args) { // 主方法
17        System.out.println( max(2, 4) ); // 传递2个实参，显示结果：4
18        System.out.println( max(2, 4, 6) ); // 传递3个实参，显示结果：6
19        System.out.println( max(2, 4, 6, 8) ); // 传递4个实参，显示结果：8
20        System.out.println( max(2) ); // 传递1个实参，显示结果：2
21        System.out.println( max() ); // 不传递实参，显示结果：0
22    } }
```



## 3.4 数组

- 二维数组
  - 存储二维表格、矩阵这样的数据集合需要使用二维数组
  - 二维数组有两个下标，第1个为行下标，第2个为列下标

- 定义二维数组

```
int iArray[ ][ ];           // 定义一个int型二维数组的引用变量iArray  
iArray = new int[2][3];    // 创建一个2行3列的int型二维数组
```

```
int iArray[ ][ ] = new int[2][3]; // 定义引用变量的同时创建数组
```

```
int iArray[ ][ ] = { { 1, 3, 5 }, { 2, 4, 6 } }; // 定义二维数组时给出初始值
```



## 3.4 数组

- 二维数组

- 理解二维数组

- 二维数组的每一行都可以看作是一个一维数组

`int a[ ][ ] = new int[2][3];` // 定义一个2行3列的二维数组a  
等价于:

`int a[ ][ ] = new int[2][ ];` // 先定义一个2行的二维数组

`a[0] = new int[3];` // 再定义第0行的一维数组，包含3个元素

`a[1] = new int[3];` // 再定义第1行的一维数组，包含3个元素

`System.out.println( a.length );` // 显示数组a的行数: 2

`System.out.println( a[0].length );` // 显示第0行的列数: 3

`System.out.println( a[1].length );` // 显示第1行的列数: 3



## 3.4 数组

- 二维数组

- 理解二维数组

- 二维数组每一行的列数可以不同

`int a[ ][ ] = new int[2][ ];` // 先定义一个2行的二维数组

`a[0] = new int[3];` // 再定义第0行的一维数组，包含3个元素

`a[1] = new int[5];` // 再定义第1行的一维数组，包含5个元素

`System.out.println( a.length );` // 显示数组a的行数： 2

`System.out.println( a[0].length );` // 显示第0行的列数： 3

`System.out.println( a[1].length );` // 显示第1行的列数： 5





## 3.4 数组

- 二维数组

- 访问二维数组元素

**数组名**[行下标][列下标]

```
int iArray[ ][ ] = new int[2][3]; // 2行3列
```

第**0**行: iArray[**0**][0]、iArray[**0**][1]、iArray[**0**][2]

第**1**行: iArray[**1**][0]、iArray[**1**][1]、iArray[**1**][2]



# 3.4 数组

## • 二维数组

例3-18 一个二维数组的Java演示程序（Array2D.java）

```
1 public class Array2D {           // 主类
2     public static void main(String[] args) { // 主方法
3         // 定义一个2行3列的二维数组a（可认为是一个矩阵），定义时初始化
4         int a[][] = {{ 1, 3, 5 }, { 2, 4, 6 }};
5         // 数组遍历：按先行后列的次序显示各数组元素的值，需使用两重循环
6         for (int m = 0; m < a.length; m++) { // 行循环：m保存行下标
7             for (int n = 0; n < a[m].length; n++) // 列循环：n保存列下标
8                 System.out.print( a[m][n] + " ");
9             System.out.println(); // 换行显示下一行数组元素
10        }
11        // 将二维数组（矩阵）a转置后保存到b中
12        int b[][] = new int[3][2]; // 定义3行2列的二维数组b
13        // 数组遍历：计算转置矩阵b
14        for (int m = 0; m < b.length; m++) { // 行循环：m保存行下标
15            for (int n = 0; n < b[m].length; n++) { // 列循环：n保存列下标
16                b[m][n] = a[n][m]; // 将a和b的行列下标互换后赋值，这就是矩阵的转置
17                System.out.print( b[m][n] + " ");
18            }
19            System.out.println(); // 换行显示下一行数组元素
20        }
21    } }
```



## 3.4 数组

- 对象数组

- 定义对象数组

Clock **c** [ ];

c = **new** Clock[**3**];

Clock **c** [ ] = **new** Clock[**3**];

c[**0**] = **new** Clock( );

c[**1**] = **new** Clock( 8, 30, 15 );

c[**2**] = **new** Clock( 12, 0, 0 );

Clock c[ ];		引用数组对象一
数组对象一	c[0]	引用Clock对象一
	c[1]	引用Clock对象二
	c[2]	引用Clock对象三
.....		
Clock对象一	int hour;	0
	int minute;	0
	int second;	0
Clock对象二	int hour;	8
	int minute;	30
	int second;	15
Clock对象三	int hour;	12
	int minute;	0
	int second;	0

- 至此，我们才最终完成对象数组c的定义和创建工作

- 通过初始化直接创建对象

Clock **c** [ ] = { **new** Clock( ), **new** Clock( 8, 30, 15 ), **new** Clock( 12, 0, 0 ) };



## 3.4 数组

- 对象数组
  - 访问对象数组  
数组名[下标]

数组名[下标].**字段名**  
数组名[下标].**方法名**( ... )

例3-19 一个对象数组的Java演示程序（ArrayObject.java）

```
1 public class ArrayObject {           // 主类
2     public static void main(String[] args) { // 主方法
3         Clock c[ ] = new Clock[6]; // 定义一个包含6个元素的钟表对象数组
4         // 遍历数组：创建钟表对象，设置并显示其时间。各钟表对象的时差为1小时
5         for (int n = 0; n < c.length; n++) {
6             c[n] = new Clock( ); // 创建钟表对象，将其引用赋值给第n个元素
7             c[n].set(n, 0, 0);    // 设置钟表对象的时间
8             c[n].show();         // 显示钟表对象的时间
9         }
10    } }
```



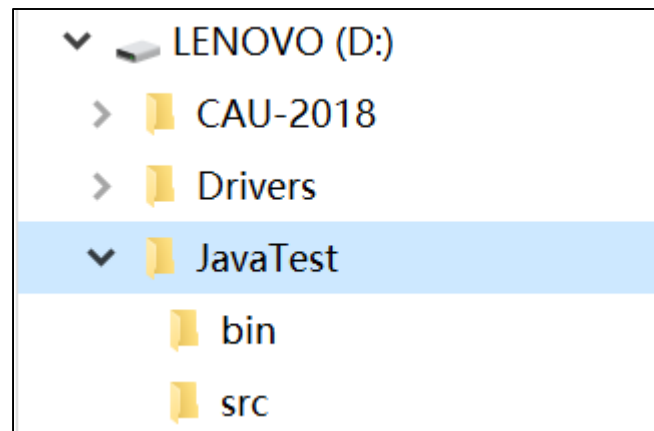
## 3.5 Java程序文件的组织

- 开发一个大型Java程序**项目**（project）可能要编写很多个**源程序文件**（source file），这就是多文件结构的Java程序。源程序文件的扩展名为“**.java**”
- 一个Java源程序文件可以包含多个**类**（class），但其中最多只能有一个**public**类，此时源程序文件的**文件名**必须与这个public类的类名相同
- 编译后，Java源程序文件中的**每个类**都会生成（或称为输出，output）一个与类同名的**类程序文件**（class file），其扩展名为“**.class**”。类程序文件所保存的是类编译之后的字节码指令



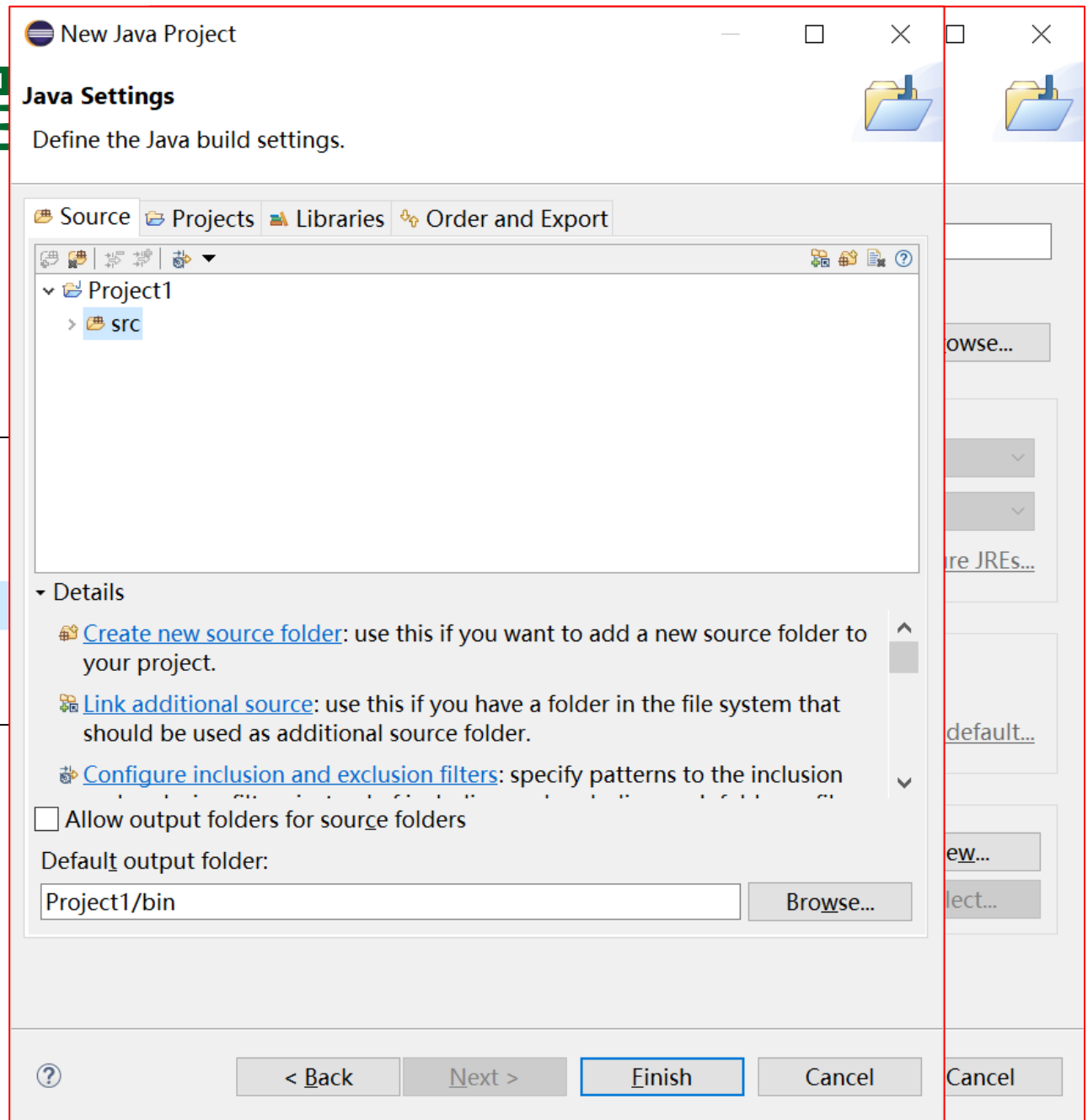
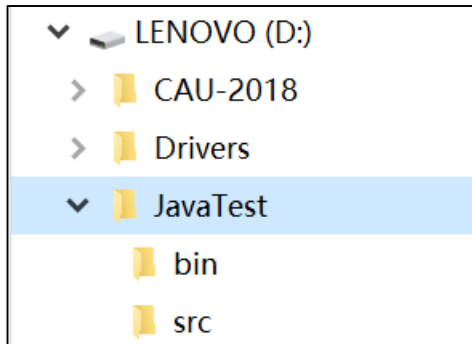
# 3.5 Java程序文件的组织

- Java项目的目录结构



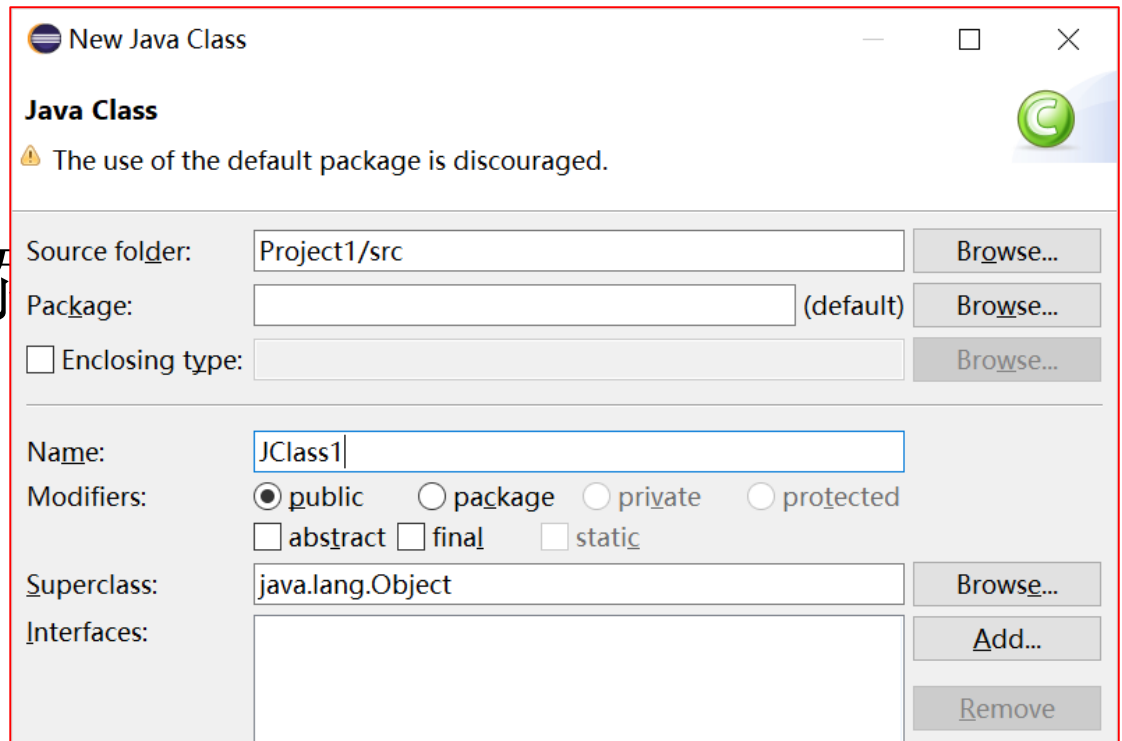
# 3.5 Java程

## • Java项目的



## 3.5 Java程序

- 在Java项目中添



New Java Class

Java Class

The use of the default package is discouraged.

Source folder: Project1/src Browse...

Package: (default) Browse...

☐ Enclosing type: Browse...

Name: JClass1

Modifiers: ☒ public ☐ package ☐ private ☐ protected  
☐ abstract ☐ final ☐ static

Superclass: java.lang.Object Browse...

Interfaces: Add... Remove

例3-20 在项目Project1中添加两个类：JClass1和JMainClass

```
1 // 类JClass1: 源程序文件JClass1.java // 主类JMainClass: 源程序文件JMainClass.java
2                                     public class JMainClass { // 主类
3 public class JClass1 {
4     private int f1 = 10; // 一个字段
5     public void show1() { // 一个方法
6         System.out.println( "JClass1: " +f1 );
7     }
8 }
```



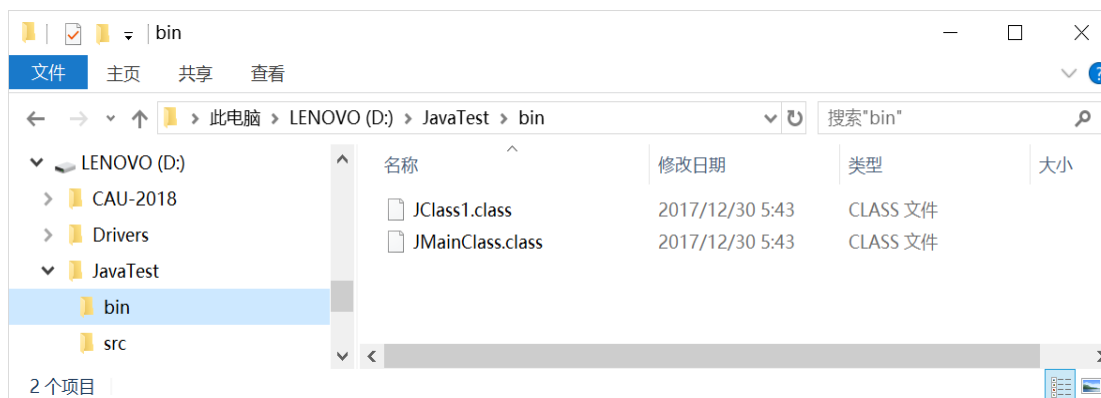
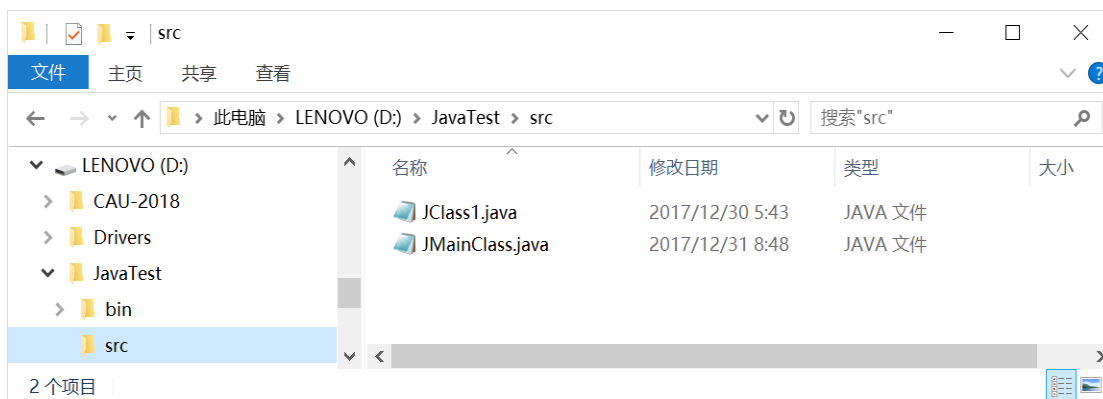
中國農業大學

閻道宏



# 3.5 Java程序文件的组织

- 在Java项目中添加Java类



中國農業大學

閻道宏

## 3.5 Java程序文件的组织

- Java源程序文件的命名细则
  - 如果文件中有一个public类，则必须以该类的类名作为文件名
  - 如果文件中没有public类，则应任选文件中某个类的类名作为文件名
  - 一个Java源程序文件中最多只能有一个public类（即访问权限被定义为public），其他类都不能指定访问权限（即访问权限是默认权限）
  - Java源程序文件的扩展名为“.java”
  - 编译后，Java源程序文件中的每个类都会生成一个与类同名的类程序文件，扩展名为“.class”



## 3.5 Java程序文件的组织

- 以包的形式管理Java类

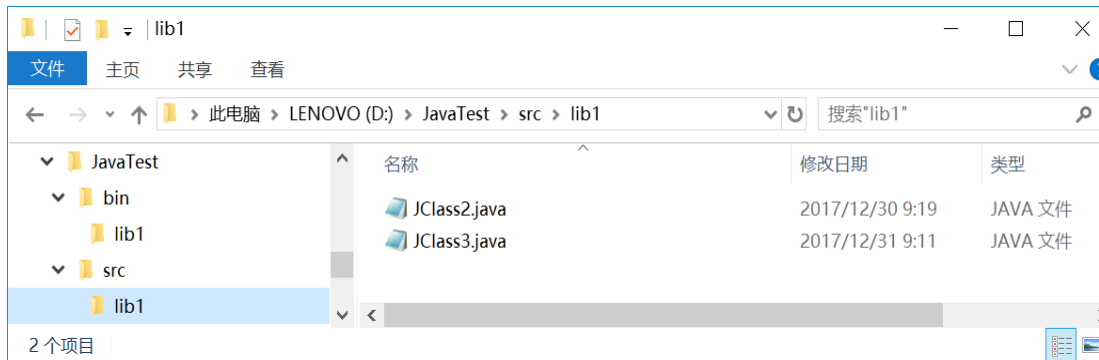
大型Java程序会定义很多个类，将会有很多个源程序文件。可以对这些源程序文件进行分组管理，即在源程序根目录src下再建立子目录，将源程序文件分散到不同的子目录下进行管理。

- 将Java源程序文件放入不同的子目录进行分组管理，实际上是对源程序文件中的类进行分组管理
- 将Java源程序文件放入不同的子目录，Java语言称之为“将文件中的类放入不同的包（package）”
- 源程序文件的子目录名被称为是类的包名



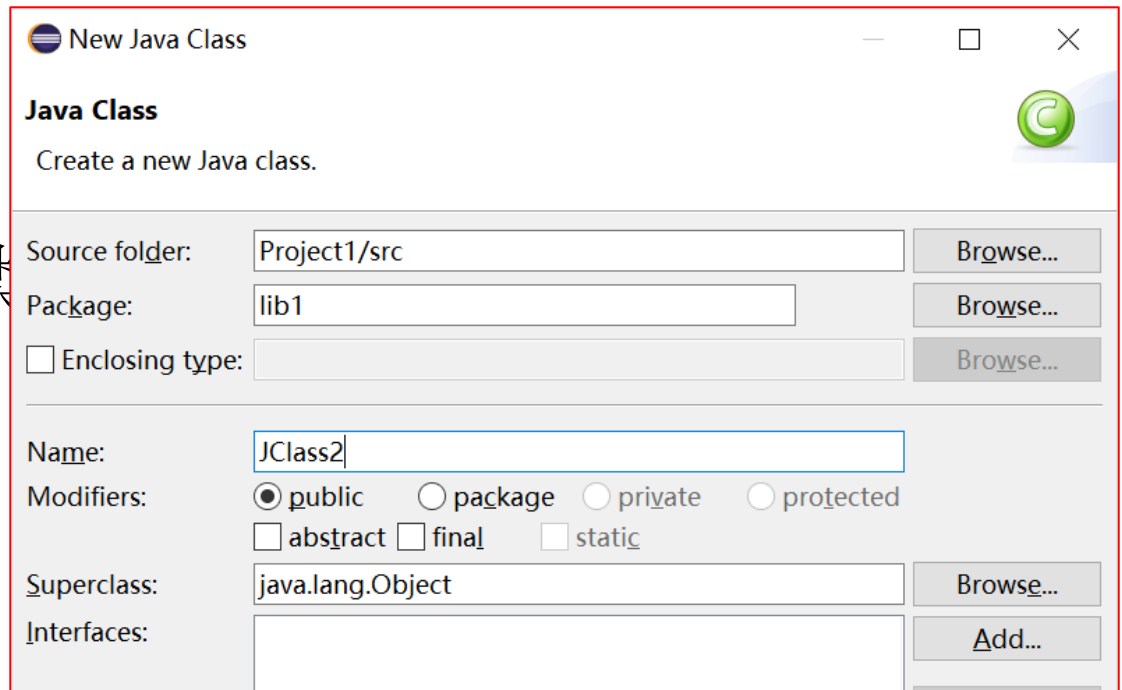
## 3.5 Java程序文件的组织

- 分包管理Java类
  - 在项目Project1（D:\JavaTest）的源程序根目录src下再建立一个子目录lib1
  - 然后在该子目录中新建两个类JClass2和JClass3



# 3.5 Java程序

## • 分包管理Java类



New Java Class

Java Class

Create a new Java class.

Source folder: Project1/src [Browse...](#)

Package: lib1 [Browse...](#)

☐ Enclosing type: [Browse...](#)

Name: JClass2

Modifiers: ☒ public ☐ package ☐ private ☐ protected  
☐ abstract ☐ final ☐ static

Superclass: java.lang.Object [Browse...](#)

Interfaces: [Add...](#)

例3-21 在项目Project1的包lib1中添加两个类：JClass2和JClass3

```
1 // 类JClass2：源程序文件lib1\JClass2.java
2 package lib1; // 向编译器告知包名
3
4 public class JClass2 {
5     private int f2 = 20; // 一个字段
6     public void show2( ) { // 一个方法
7         System.out.println( "JClass2: " +f2 );
8     }
9 }
```

```
// 类JClass3：源程序文件lib1\JClass3.java
package lib1; // 向编译器告知包名

public class JClass3 {
    private int f3 = 50; // 一个字段
    public void show3( ) { // 一个方法
        System.out.println( "JClass3: " +f3 );
    }
}
```



中國農業大學

閻道宏

# 3.5 Java程序文件的组织

## • 分包管理Java类

Java语法：package语句

**package** 包名;

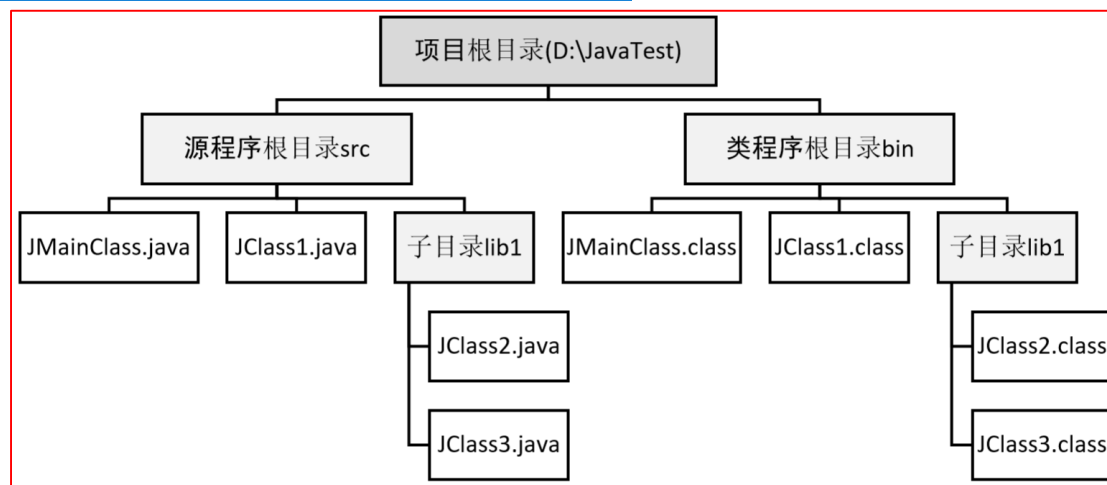
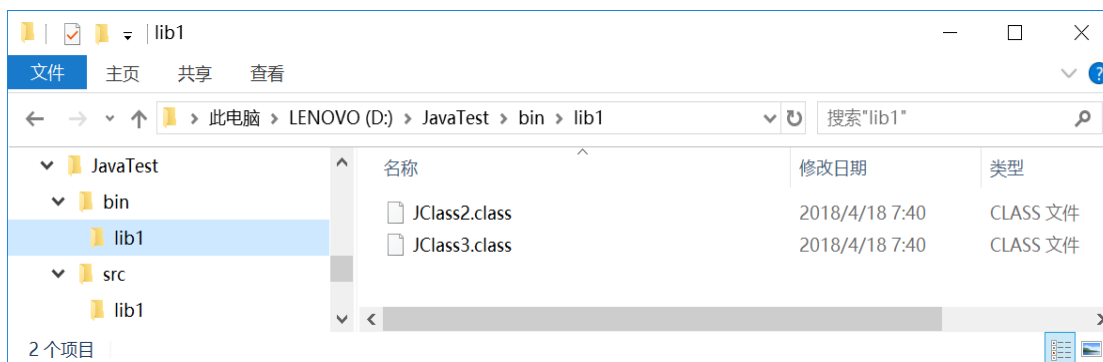
语法说明：

- **package**语句的作用是告知编译器本文件中类所在的包名，它应当是源程序代码的第一条语句（注释除外）。**package**是Java语言的关键字。
- **包名**指定了源程序文件（.java）所在的子目录名，该子目录名是在源程序根目录下的相对路径名。这时，我们称“源程序文件（.java）中的类被放入到了指定的包中”。
- 编译时，Java编译器会按照**package**语句在类程序根目录下创建完全相同的子目录结构，并将编译生成的类程序文件（.class）自动放入对应的子目录中。
- 存放在源程序根目录下的类不需要添加**package**语句。Java称“这些类被放入了默认（**default**）包（或称无名包）中”。
- 可以在源程序根目录下建立多级子目录。这时，包名的命名形式为“一级子目录.二级子目录.....”子目录之间用点“.”隔开。
- **包名**（同时也是类所在的子目录名）习惯上以小写字母开头，**类名**（同时也是类的程序文件名）习惯上以大写字母开头，这样可以很容易辨识出包名与类名（即子目录名与文件名）。



# 3.5 Java程序文件的组织

- 分包管理Java类

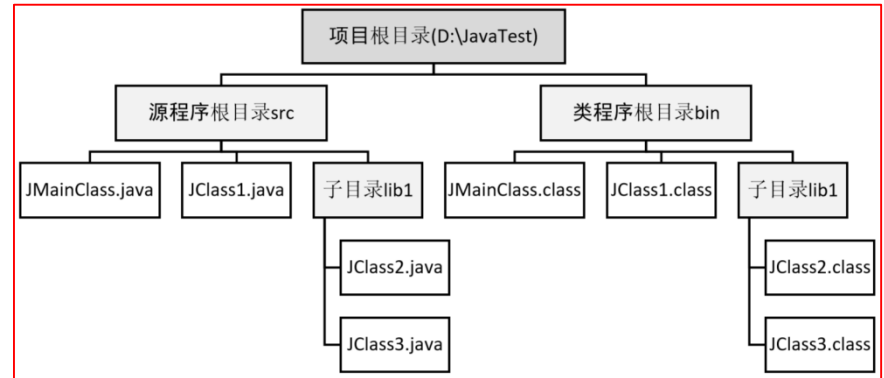


# 3.5 Java程序文件的组织

- 使用不同包里的类

- JMainClass

- JClass1
- JClass2和JClass3



- 使用同一包中的类时，直接使用类名

**JClass1** obj1 = **new JClass1**( );

- 使用不同包里的类，需在类名之前加上包名

lib1.**JClass2** obj2 = **new lib1.JClass2**( );

lib1.**JClass3** obj3 = **new lib1.JClass3**( );

- 使用不同包里的类可以预先导入，然后直接使用类名





## 3.5 Java程序文件的组织

- 使用不同包里的类

例3-22 一个使用不同包里类的Java演示程序（JMainClass.java）

```
1 import lib1.JClass2; // 预先导入包lib1中的类JClass2
2 import lib1.JClass3; // 预先导入包lib1中的类JClass3
3 // import lib1.*; // 或者预先导入包lib1中的所有类
4
5 public class JMainClass {           // 主类
6     public static void main(String[] args) { // 主方法
7         JClass1 obj1 = new JClass1(); // 使用同一包里的类JClass1，直接使用类名
8         obj1.show1();
9         // 下面使用lib1包里的类JClass2、JClass3
10        JClass2 obj2 = new JClass2(); // 因为预先导入了lib1包里的类，这里可直接使用类名
11        JClass3 obj3 = new JClass3();
12        /* 如果不预先导入lib1包，则需在类名之前加上包名。例如，
13        lib1.JClass2 obj2 = new lib1.JClass2( ); // 使用lib1包里的类，需在类名前加上包名
14        lib1.JClass3 obj3 = new lib1.JClass3( );
15        */
16        obj2.show2(); obj3.show3();
17    } }
```



## 3.5 Java程序文件的组织

- 使用不同包里的类

Java语法: import语句

**import** 包名.类名;

**import** 包名.\*;

语法说明:

- **import**语句应放在源程序的开头（仅次于package语句）。import是Java语言的关键字。
- “包名.类名”指定导入包中的某个类；“包名.\*”则是导入包中的所有类，但不包括其子包（即下级子目录）中的类。
- 后续程序使用被导入的类，可直接使用类名，这样可以简化程序代码；导入后仍可以继续“包名.类名”的形式。
- 如果从不同包中导入了多个重名的类，此时必须使用“包名.类名”的形式，因为只有这样才能明确指定使用哪个包中的类。
- 注：Java语言中包的作用类似于C++语言中的命名空间（namespace），但Java里的包会实际对应文件系统中的目录（文件夹），而C++里的命名空间则不会。



中國農業大學

閻道宏

## 3.5 Java程序文件的组织

- 使用不同包里的类

- 导入类中的静态成员

- 举例：访问数学类Math中静态成员

```
System.out.println( Math.PI );
```

```
System.out.println( Math.sqrt( 8.6 ) );
```

- 可以预先导入类中的静态成员，访问时省略类名，直接使用静态成员名

```
import static Math.*; // 先导入数学类Math里的所有静态成员
```

```
System.out.println( PI ); // 访问静态字段PI时可省略类名
```

```
System.out.println( sqrt( 8.6 ) ); // 调用静态方法sqrt时可省略类名
```



中國農業大學

閻道宏

## 3.5 Java程序文件的组织

- 使用不同包里的类
  - 导入类中的静态成员

Java语法: import static语句

```
import static 包名.类名.静态成员名;  
import static 包名.类名.*;
```

语法说明:

- **import static**语句用于导入某个类中的静态成员。其中, “包名.类名.\*”表示导入类中的所有静态成员。“import static”语句可称为静态导入语句。
- 通常, 访问类中静态成员的形式应当是“包名.类名.静态成员名”。导入后可省略“包名.类名.”, 直接使用静态成员名访问, 这样可以简化程序代码。
- 导入后仍可以继续使用“包名.类名.静态成员名”的形式进行访问。

### – 包的绝对路径

- **Java虚拟机**

- 运行Java虚拟机时选项“-classpath”所指定的目录
- 环境变量CLASSPATH所指定的目录

- **Eclipse**: Java项目的组建路径 (Java Build Path)



中國農業大學

閻道宏

# 3.5 Java程序文件的组织

- 访问权限
  - Java语言中的访问权限控制分为两级
    - 第一级先设定类的访问权限
    - 第二级再设定类中成员的访问权限
  - 类的访问权限
    - 公有权限**public**。具有公有权限的类是开放的。使用**public**类不受控制
    - **默认权限**：定义时没有为类设定访问权限，则该类具有默认权限。如果定义类时未指定访问权限，这意味该类只能被
      - 同一程序文件的类
      - 同一目录下其他程序文件中的类使用
      - 默认权限是向本文件或本包定向开放的一种权限



## 3.5 Java程序文件的组织

- 访问权限
  - 类成员的访问权限
    - 公有权限、私有权限、默认权限和保护权限
  - 公有权限**public**。具有公有权限的类成员是开放的。访问**public**成员不受控制
  - 私有权限**private**。具有私有权限的类成员是隐藏的。**private**成员只能在本类中访问，即只能被本类成员访问
  - **默认权限**：定义时没有为类成员设定访问权限，则该类成员具有默认权限。如果定义类成员时未指定访问权限，这意味着该类成员只能被
    - 同一程序文件的类
    - 同一目录下其他程序文件中的类访问
  - 保护权限**protected**



## 3.5 Java程序文件的组织

- JAR包
    - 可以将编译好的类程序文件 (\*.class) 压缩打包成Java归档文件 (Java ARchive, 简称为**JAR**)
    - 使用JDK中的归档打包程序 (**jar.exe**) 对类程序文件及其附属文件 (例如图片文件) 进行压缩打包, 所生成的归档文件扩展名为 “**.jar**”。
- Java归档文件俗称为 “**JAR包**”



## 3.5 Java程序文件的组织

- JAR包
  - 与JAR包相关的4条常用命令
    - 创建、查看、解压或运行JAR包

表3-2 与JAR包相关的4条命令

功能	命令
创建JAR包	<code>jar cf jar-file input-file(s)</code>
查看JAR包中的内容	<code>jar tf jar-file</code>
解压JAR包，提取文件	<code>jar xf jar-file</code>
运行JAR包中的主类	<code>java -jar app.jar</code>



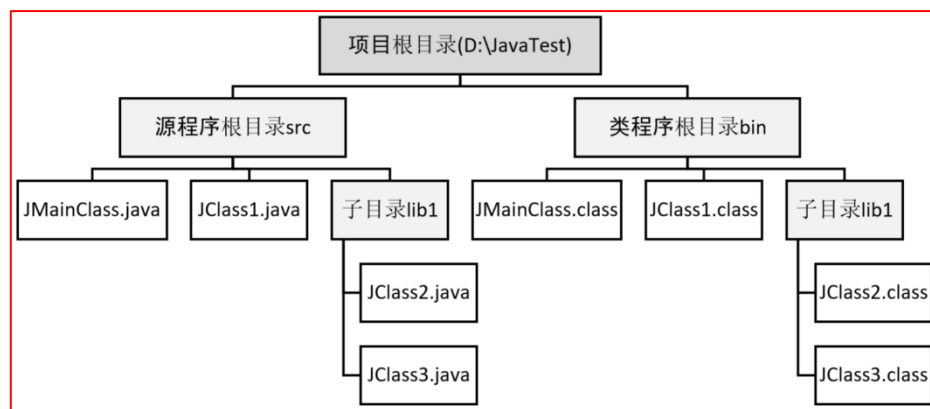


# 3.5 Java程序文件的组织

- JAR包

- **cd** d:\JavaTest\bin <回车>

- 仅打包一个类程序文件



```
C:\WINDOWS\system32\cmd.exe

D:\JavaTest\bin>jar cf myLib.jar JClass1.class

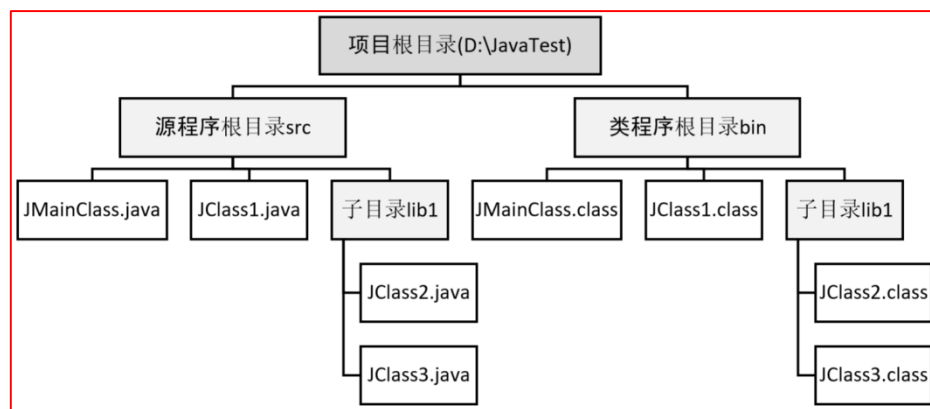
D:\JavaTest\bin>jar tf myLib.jar
META-INF/
META-INF/MANIFEST.MF
JClass1.class

D:\JavaTest\bin>_
```



# 3.5 Java程序文件的组织

- JAR包
  - 打包一个子目录



```
C:\WINDOWS\system32\cmd.exe

D:\JavaTest\bin>jar cf myLib1.jar lib1\*.class

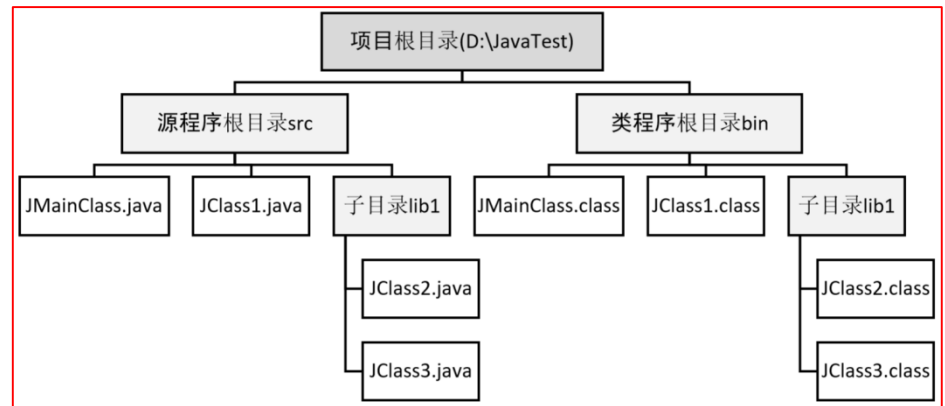
D:\JavaTest\bin>jar tf myLib1.jar
META-INF/
META-INF/MANIFEST.MF
lib1/JClass2.class
lib1/JClass3.class

D:\JavaTest\bin>
```



# 3.5 Java程序文件的组织

- JAR包
  - 打包一个可执行JAR包
  - 清单（manifest）文件
    - Manifest-Version: 1.0
    - Main-Class: JMainClass



```
C:\WINDOWS\system32\cmd.exe

D:\JavaTest\bin>jar cfm myApp.jar Manifest.txt *.class lib1\*.class

D:\JavaTest\bin>jar tf myApp.jar
META-INF/
META-INF/MANIFEST.MF
JClass1.class
JMainClass.class
lib1/JClass2.class
lib1/JClass3.class

D:\JavaTest\bin>java -jar myApp.jar
JClass1: 10
JClass2: 20
JClass3: 50

D:\JavaTest\bin>
```

# 第3章 面向对象程序设计之一

- 本章学习要点

- 深入理解**面向对象程序设计**方法的基本原理和设计过程
- 掌握Java语言中类与对象的**语法规则**
- 理解**引用数据类型**与基本数据类型之间的区别
- 掌握Java语言中**数组**相关的语法
- 掌握Java语言多文件结构的管理方法，重点理解**包**和**子目录**之间的对应关系

