JAVA程序设计

第3章

Class and Object

内容提要

- 面向对象程序设计概要
- 3. 1 Java类
- 3. 2 Java对象
- 3. 3 包：库单元
- 3. 4 Java标准类库

面向对象的程序设计

◆程序设计思想：结构化与面向对象

➢程序设计思想和理论的发展

- 传统的面向过程程序设计：数据与程序分离
- 现代的面向对象程序设计：数据与操作封装在类中

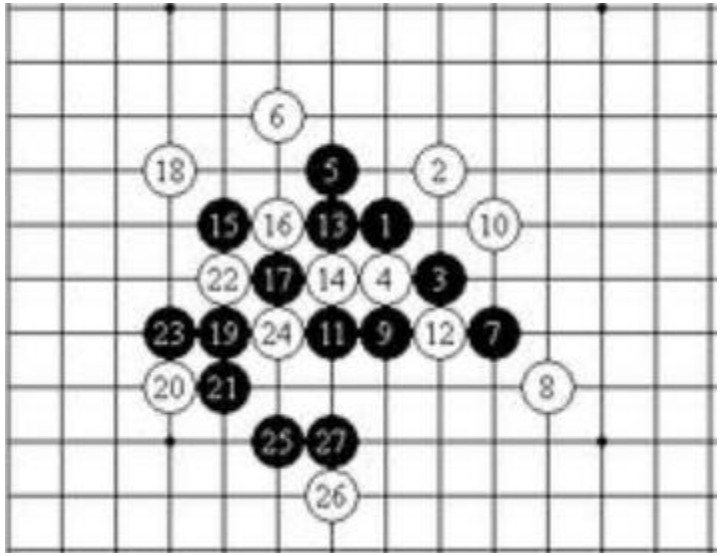
➢面向对象程序设计方法的好处

- 化大程序为小程序，改善程序的可维护性
- 提高软件代码的重用性，避免重复开发

面向对象的程序设计

面向过程程序设计：分析出解决问题的步骤，用函数把这些步骤一步一步实现，使用时依次调用。

面向对象程序设计：把构成问题事物分解成各个对象，建立对象不是为了完成某个步骤，而是为了描述某个事物在整个解决问题步骤中的行为。



面向过程分析
1：开始游戏
2：黑子先走
3：绘制画面
4：判断输赢 yes 跳转8
5：轮到白子
6：绘制画面
7：判断输赢 no 跳转2
8：输出结果

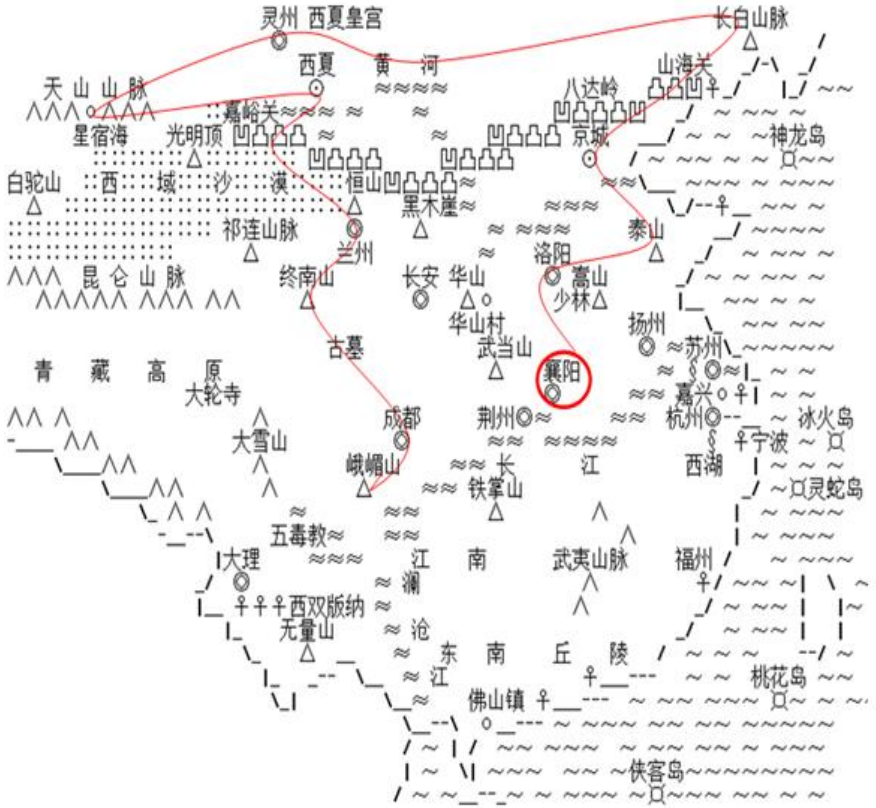
面向对象分析
1：player
2：棋盘系统
3：规则系统

面向对象的程序设计


KBA
Google DeepMind
Challenge Match
8 - 15 March 2016
AlphaGo
Google DeepMind
Challenge Match

面向对象的程序设计

◆程序设计思想：结构化与面向对象

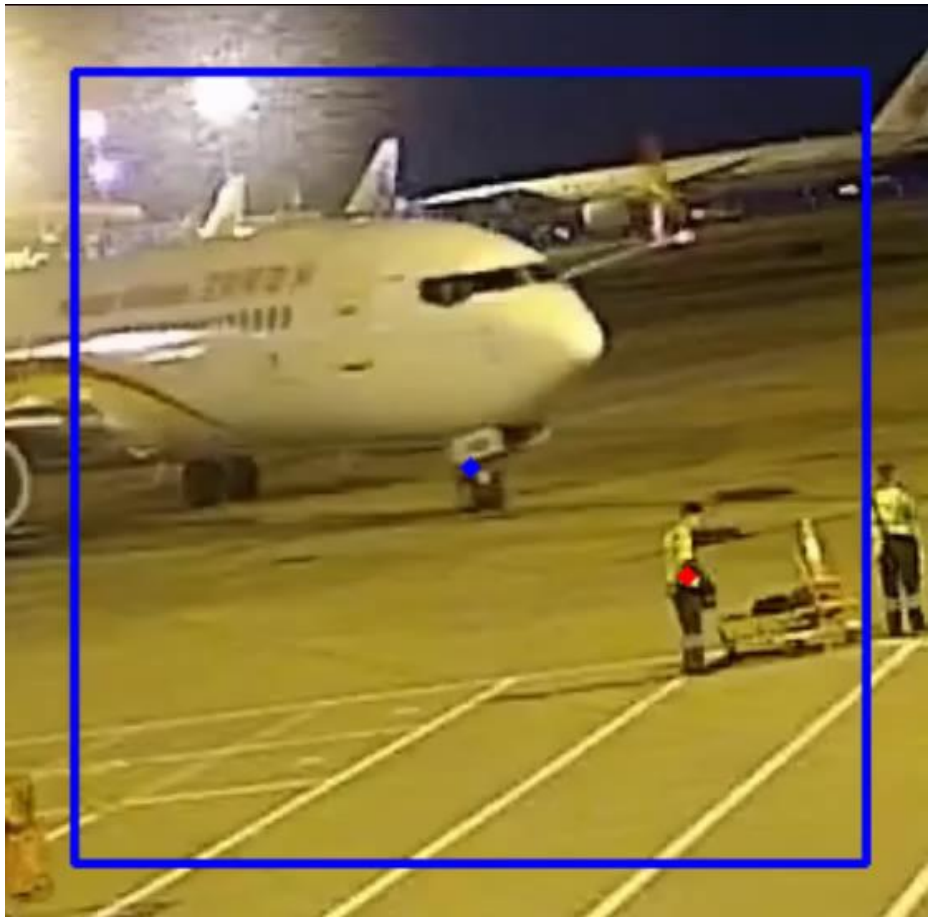


面向过程

面向对象

面向对象的程序设计

- 程序设计思想：结构化与面向对象



面向对象的程序设计

- 面向对象的三大特性：
  - 封装性（**Encapsulation**）
  - 继承性（**Inheritance**）
  - 多态性（**Polymorphism**）

优势？好处？

面向对象的程序设计

◆封装：对象、类和消息

- 一个**对象**就是变量和相关的方法的集合，面向对象程序设计实现了对象的封装，实现了模块化和信息隐藏，有利于程序的可移植性和安全性。
- **类**是若干对象所具有的共性。在类中定义所有对象所共有的内容，将其实例化即生成了对象。
- **消息**是对象之间的交互方式和交互内容。消息包括：

✓消息的接收者

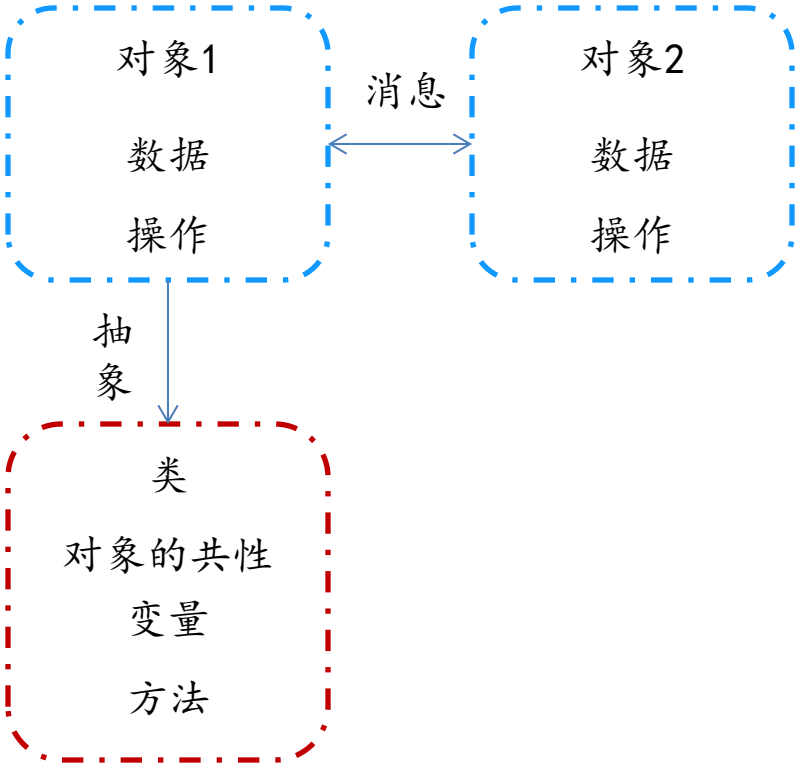
✓接收对象应采用的应对方法

✓方法所需要的参数

封装的好处？

面向对象的程序设计

◆封装：对象、类和消息



[例1.1]用Java语言程序在屏幕上显示一条语句“this s a simple program!”

```
// Example 1 of Chapter 1
//This is a simple instance of java application
public class SimpleApp
{
        public static void main(String args[])
        {
                System.out.println("this is a simple program!");
        }
}
```

注释行

主类

主方法
程序入口

应用类

结束主方法

结束主类

3.1 Java类

- 3.1.1类的定义
- 3.1.2变量成员与方法成员
- 3.1.3访问控制符
- 3.1.4构造方法
- 3.1.5终结处理方法finalize

5.数据类型(Data Types)

- 数据
  - 基本数据
    - 数值型
      - 整数类型（byte，short，int，long）
      - 浮点类型（float，double）
    - 字符类型（char）
    - 逻辑类型（boolean）
  - 引用数据
    - 类（class）
    - 接口（interface）
    - 数组

3.1 Java类

◆3.1.1类的定义

- 类是组成Java程序的基本要素，既是组织程序代码的基本单位，也是面向对象程序设计的主体。

- 类声明形式：

```
[public][abstract|final] class  classname
{
        classbody
}
```

- 例子：

```
class ATypeName {/*类主体置于这里*/}
ATypeName a = new ATypeName();
```

3.1 Java类

◆3.1.2变量成员与方法成员

- 在Java世界中，我们全部的工作就是**定义类**、制作这些**类的对象**以及将**信息发给**这些对象。
- 类里设置两种类型的元素：“**变量成员**”与“**方法成员**”。

```
public class classname {
//定义变量成员
//定义方法成员
}
```

◆3.1.2变量成员与方法成员

➢变量成员的基本声明格式为

[public|protected|private][static][final][transient][volatile] type variableName;

- 变量成员的类型可以是基本数据类型，也可以是引用数据类型。
- 其名字在类成员中必须是唯一的，不能与其他变量同名，但可以和类内的某个方法使用同一个名字。
- 变量成员的作用域为整个类。

3.1 Java类

◆3.1.2变量成员与方法成员

- ➢一个只包含变量成员类的示例

```
class DataOnly {
    int i;
    float f;
    boolean b;}
DataOnly d = new DataOnly();
```

- ➢变量成员可以是引用类型

```
class DataReference {
    int j;
    DataOnly d;}
 DataReference dr= new DataReference();
```

◆3.1.2变量成员与方法成员

➢变量成员的访问方式

objectReference.variableName

```
d.i = 47;
d.f = 1.1f;
d.b = false;

dr.j = 23;
dr.d = new DataOnly();
dr.d.i = 100;
```

◆3.1.2变量成员与方法成员

➢方法成员的基本声明格式为

[public|protected|private][static] returnType

methodName([paramList]){methodbody}

●其中returnType可以是基本数据类型，也可以是引用数据类型，或者当无返回值时为void。

●参数列表中的参数也可以是各种数据类型。

●方法体是方法的实现，其中可以包含局部变量的声明和所有合法的Java语句。事实上这是Java语言程序的实质部分，程序中所有的执行动作都是在这里实现的。

◆3.1.2变量成员与方法成员

●如果方法的返回类型不为void型，则在方法体中必须包含return语句，且return语句必须被执行一次并且只能执行一次，并返回一个与返回类型相同的值。

●如果方法的返回类型为void型，则在方法体中不必包含return语句。

●return语句在方法中的使用方式为：

return expression; 或者是用return;方式只返回控制。

◆3.1.2变量成员与方法成员

➢方法成员的访问方式

●在Java中，方法只能作为类的一部分来创建，方法通过对象调用，且这个对象必须能够执行这个方法调用。

```
ojbectName.methodName(arg1, arg2, arg3);
int x = a.f();
```

●这种调用方法的行为通常被称为发送消息给对象。

◆3.1.2变量成员与方法成员

➢static成员

●创建类时，就是在描述这个类的对象的属性与行为。除非用new创建这个类的对象，否则并未获得任何对象。

●两种情形：为某特定域分配单一存储空间，而不去考虑究竟要创建多少对象，甚至根本就不创建任何对象；某个方法不与包含它的类的任何对象关联在一起。

◆3.1.2变量成员与方法成员

➢static成员：类变量成员和类方法成员

●在类定义中可以用static修饰其变量成员和方法成员，使其具有static属性，其被访问方式和访问范围将发生一些变化。

●格式：

```
static type classVariable
static returnType classMethod([paramList]){…}
```

●用static声明的变量成员为类变量，不用static声明的变量成员为实例变量。

◆3.1.2变量成员与方法成员

➢Static成员：类变量和实例变量

●类变量由多个对象实例共享一个内存区，共享同一个值。实例变量各自有各自的内存区，对于不同的对象实例而言，其对应的变量成员有不同的值。

●实例变量必须在生成对象实例后通过对象实例名来访问；类变量既可以通过对象实例名访问，也可以通过类名直接访问。

◆3.1.2变量成员与方法成员

➢static成员：类变量和实例变量举例

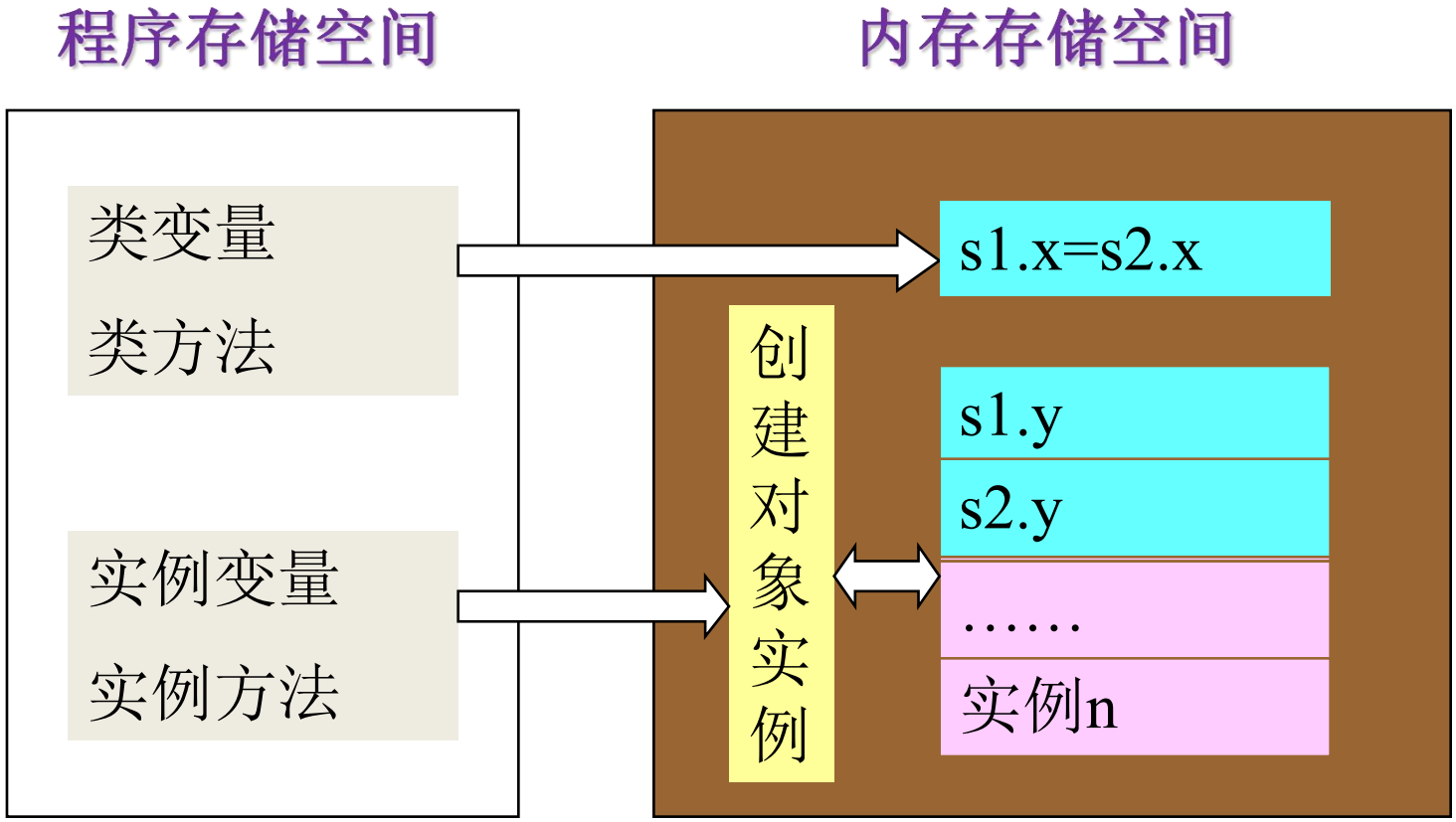
```
\\StaticVarTest.java
public class StaticVarTest {
    static int x=100;
    int y=1;
    public StaticVarTest(){}
}
```

```
\\StaicVarTestDemo.java
public class StaticVarTestDemo {
    public static void main(String args[])
    {
      StaticVarTest s1=new StaticVarTest();
      StaticVarTest s2=new StaticVarTest();
      System.out.println("修改x之前，s1.x="+s1.x+"    s2.x="+s2.x);
      s1.x++;
      System.out.println("修改x之后，s1.x="+s1.x+"    s2.x="+s2.x);
      System.out.println("修改y之前，s1.y="+s1.y+"    s2.y="+s2.y);
      s1.y++;
      System.out.println("修改y之后，s1.y="+s1.y+"    s2.y="+s2.y);
    }
}
```

```
修改x之前，s1.x=100   s2.x=100
修改x之后，s1.x=101   s2.x=101
修改y之前，s1.y=1   s2.y=1
修改y之后，s1.y=2   s2.y=1
```

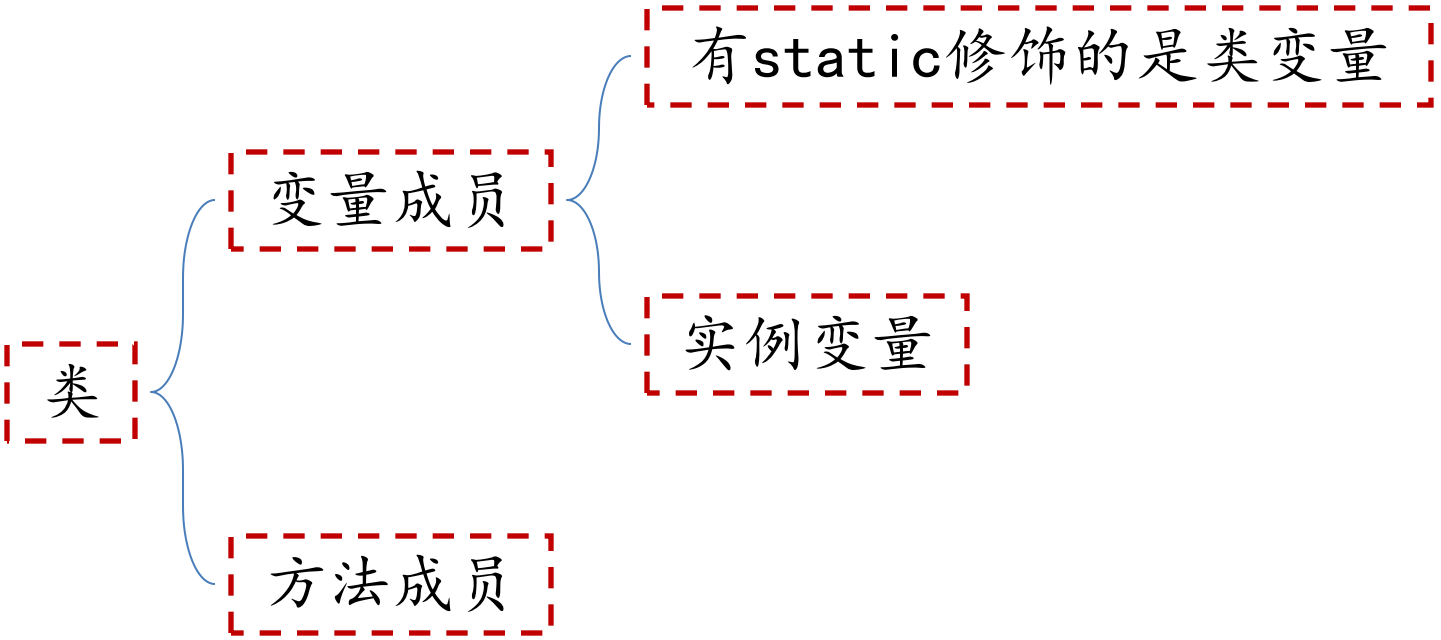
◆3.1.2变量成员与方法成员

➢static成员：示意图



3.1 Java类

◆3.1.2变量成员与方法成员



◆3.1.2变量成员与方法成员

➢static成员：类方法和实例方法

●用static声明的方法成员为类方法，不用static声明的方法成员为实例方法。

●在类方法中不能使用this或super。

●可以不生成类对象而直接通过类名访问类变量和类方法。这就是为什么将其称为“类”成员。

```
class Incrementable {
    static int i = 1;
    static void increment() {i++;}
}
Incrementable.increment();
```

◆3.1.2变量成员与方法成员

```
public class SoftwareNumber
{
    private int serialNumber;                          //实例变量
    public static int counter=0;                  //类变量
    public SoftwareNumber()                            //实例方法
    {
          counter++;                                   //实例方法中访问类变量
          serialNumber=counter;
    }
    public static int getTotalNumber()          //类方法
    {
          return counter;                              //类方法中访问类变量
    }
    public int getSerialNumber()                       //实例方法
    {
          return serialNumber;                   //实例方法中访问实例变量
    }
}
```

不允许类方法 访问实例变量！

◆3.1.2变量成员与方法成员

➢声明的作用域

Java语言的作用域分为类级、方法级、语句块级、语句级。

●在类体中声明的变量成员和方法成员的作用域是整个类。

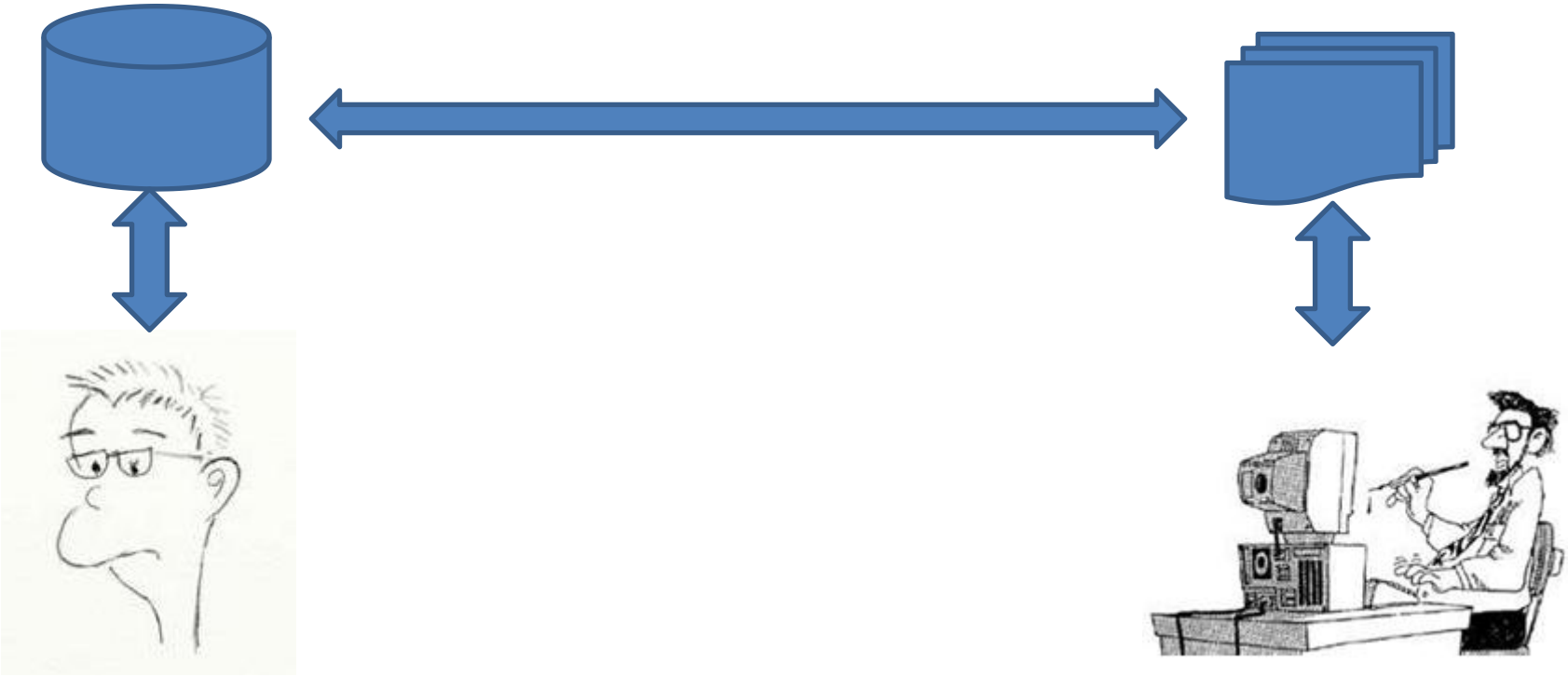
●在方法中声明的参数和在方法中所有语句之外声明的变量的作用域是整个方法体。

●在语句块中声明的局部变量的作用域是该语句块。

●在语句中声明的变量的作用域是该语句。

3.1 Java类

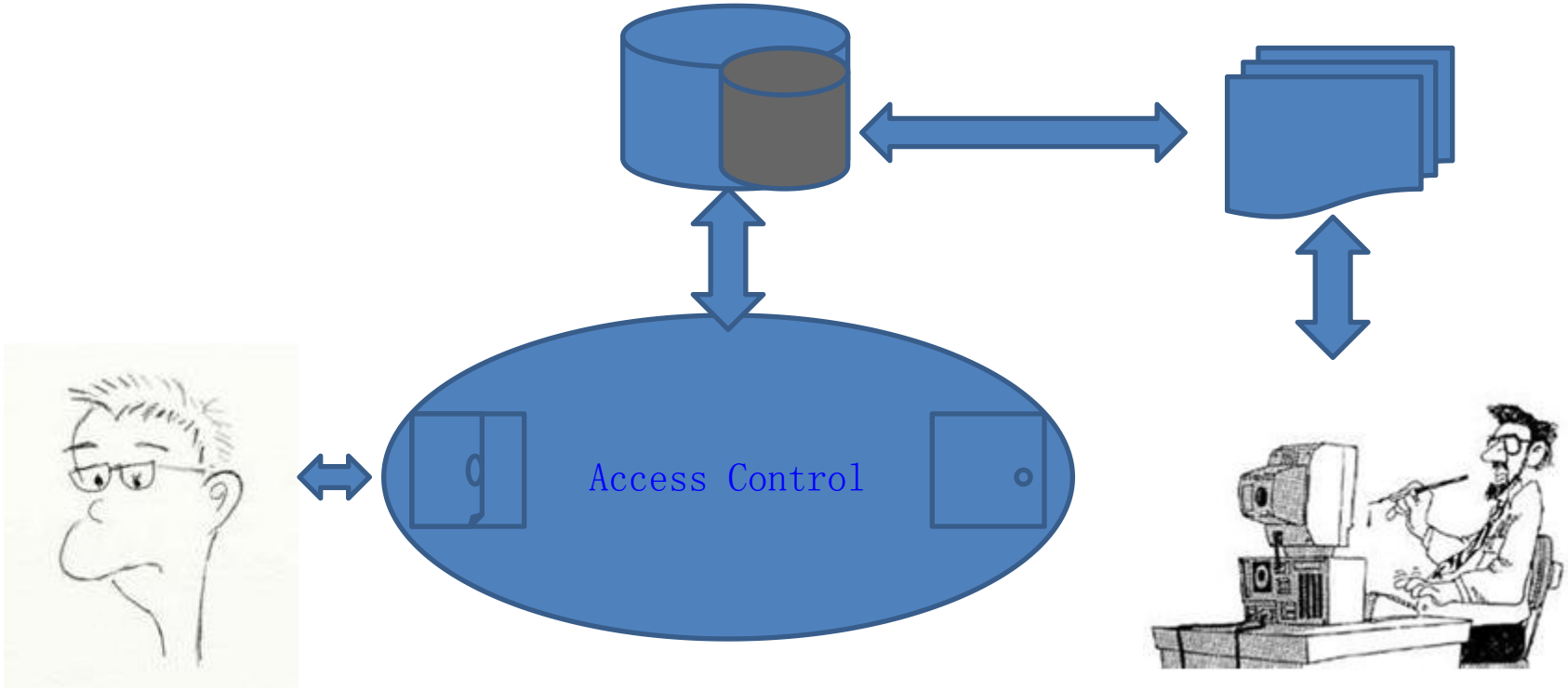
◆3.1.3访问控制符Access Control



“将那些保持不变与发生变化的事物区分开来.”

3.1 Java类

◆3.1.3访问控制符Access Control



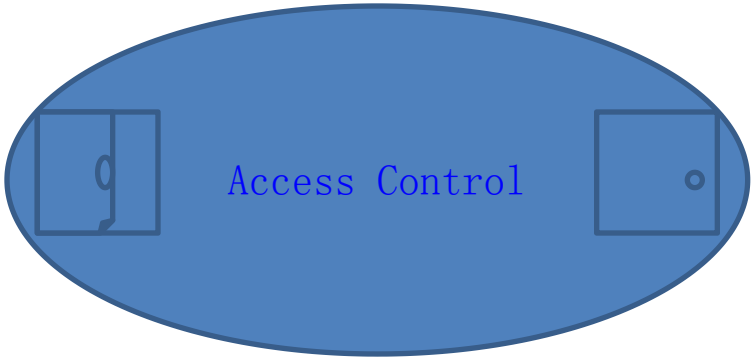
“将那些保持不变与发生变化的事物区分开来.”

◆3.1.3访问控制符Access Control

- 可以对所有的成员分别设定访问权限，以限定其他对象对它的访问，Java语言设定以下四种访问权限：
- 包可以用来组织类。

```
package packageName;
```

public
protected
friendly(包访问权限)
private



3.1 Java类

◆3.1.3访问控制符

- public: 使用关键字public，就意味着public之后紧跟着的成员，对每个人都是可用的，可以被任何其他类的对象访问。
- protected: protected控制符修饰的成员能在它自己的类中和继承它的子类中被访问，也可以被同一包中的类访问。
- 默认： 也叫包访问权限，友元friendly。如果没有使用可见性的控制符，那么默认为类、方法和变量是可以被同一个包中任何一个类访问。
- private: private控制符修饰的成员只能在它自己的类中被访问。

3.1 Java类

◆3.1.3访问控制符

	同一个类中	同一个包中	不同包中的子类	不同包中的非子类
public	√	√	√	√
protected	√	√	√	
默认	√	√		
private	√			

3.1 Java类

◆3.1.3访问控制符

```
package p1;
public class C1{
    public int x;
    int y;
    protected int z;
    private int w;

    public void m1(){
    }
    void m2(){
    }
    protected m3(){
    }
    private void m4(){
    }
}
```

```
package p1;
public class c2{
    void aMethod(){
        C1 o = new C1();
        int a = o.x;  //public权限，可以访问
        int b = o.y;  //包访问权限，可以访问
        int c = o.w;  //private权限，不可以访问
    }
}
```

```
package p2;
public class c3{
    void aMethod(){
        C1 o = new C1();
        int a = o.x;  //public权限，可以访问
        int b = o.y;  //包访问权限，不可以访问
        int c = o.w;  //private权限，不可以访问
    }
}
```

◆3.1.4构造方法(Construction Method)

●随着计算机革命的发展，不安全的编程方式已逐渐成为编程代价高昂的主要原因之一。初始化和清理正是涉及安全的两个问题。

●在Java 中，通过提供构造方法，类的设计者可确保每个对象都会得到初始化。

◆3.1.4构造方法(Construction Method)

●构造方法用来**初始化类对象**。

●构造方法是一个特殊的方法，方法名与**类名相同**，并且无返回类型。

●如果在定义类时没有定义构造方法，则Java系统会自动提供**默认的**无参数构造方法。所以，**任何类**都有构造方法。

- 构造方法可以重载。
- 构造方法只能用new关键字调用。
- 若构造方法体定义为空，则自动调动其父类的构造方法。

◆3.1.4构造方法

```
class Rock{
    Rock(){
        System.out.print(“Rock ”);
    }
}
public class SimpleConstructor{
    public static void main(String[] args){
        for(int i =0; I < 10; i++)
            new Rock();
     }
}/* output
Rock Rock Rock Rock Rock Rock Rock
Rock Rock Rock
*/
```

```
class Rock2{
    Rock2(int i) {
        System.out.print(“Rock” + i + “ ”);
    }
}
public class SimpleConstructor{
    public static void main(String[] args){
        for(int i =0; i < 10; i++)
            new Rock2(i);
    }
}/* output
Rock 0 Rock 1 Rock 2 Rock 3 Rock 4 Rock 5
Rock 6 Rock 7 Rock 8 Rock 9
*/
```

◆3.1.5终结处理方法finalize()

- 负责回收无用对象占据的特殊的内存资源。由于在分配内存时可能采用了类似C语言中的做法，而非Java中的通常做法，通过某种创建对象方式以外的方式为对象分配了存储空间，即并非由new所创建的一块特殊的内存区域。
- finalize()不是C++中的析构函数，不能将finalize()作为通用的清理方法。

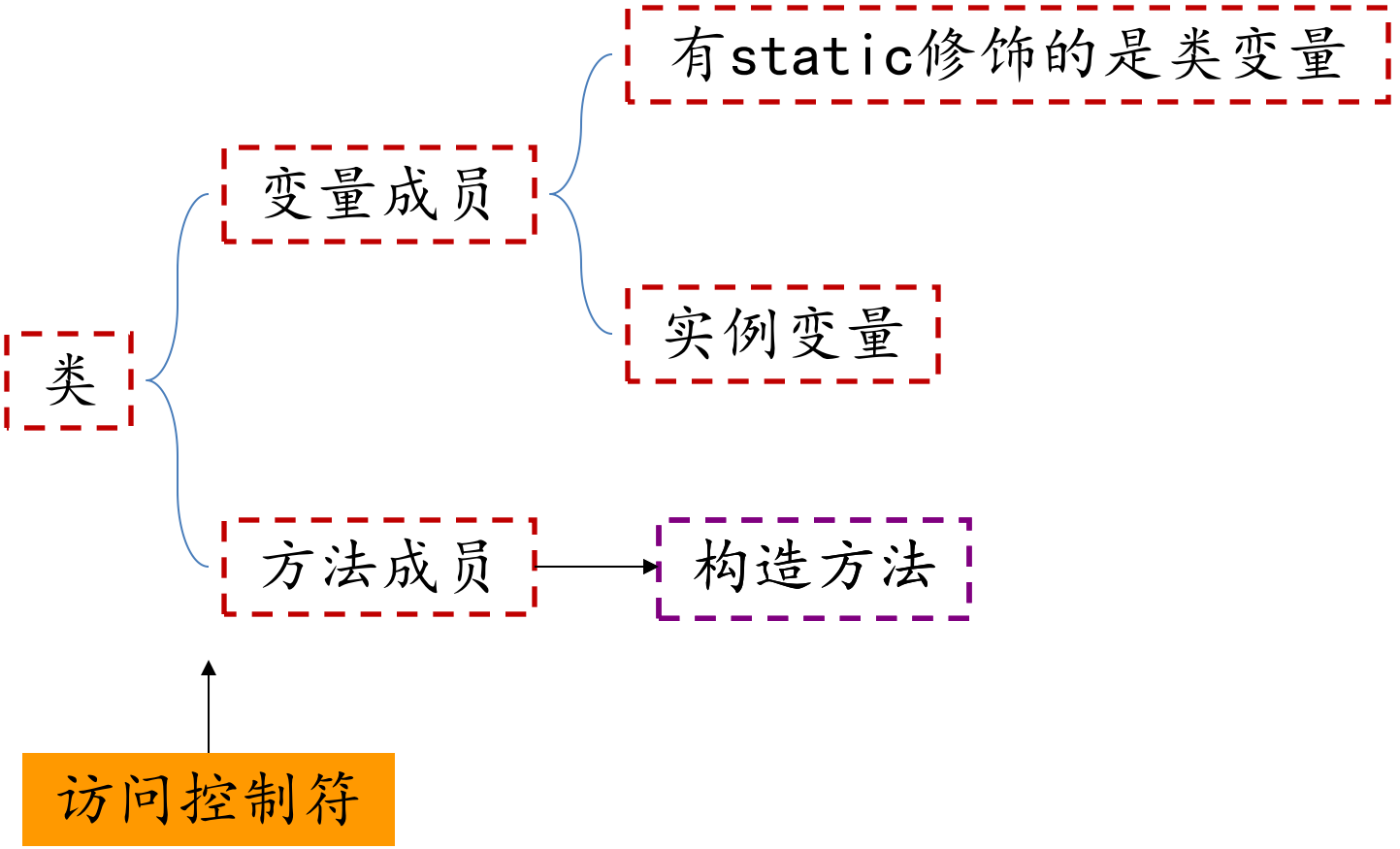
3.1 Java类

◆小结：

- ➢面向对象程序设计思想
- ➢对封装的理解，Java类是对象的抽象、实现与使用相分离
- ➢变量成员与方法成员的声明方式与调用方法
- ➢静态成员的特殊性
- ➢构造方法与默认构造方法

3.1 Java类

◆变量成员与方法成员



3.2 Java **Object**

- 3.2.1对象的创建
- 3.2.2对象的初始化
- 3.2.3成员初始化顺序
- 3.2.4垃圾回收机制

3.2 Java对象

◆3.2.1对象的创建

●Java中一切都是**对象**，类是创建对象的**模板**。

●对象是类在程序中的实例化。对象实例在程序中包括生成、使用和清除三个阶段。

●对象实例的生成包括声明、实例化和初始化三个步骤

```
type  objectName = new  type([paramList])
```

●这里的type objectName声明了一个对象实例的引用，new运算符为其分配内存空间，完成了实例化，type构造方法执行初始化工作。

```
Television tv;
Television tv = new Television();
```

3.2 Java对象

◆3.2.1对象的使用与清除

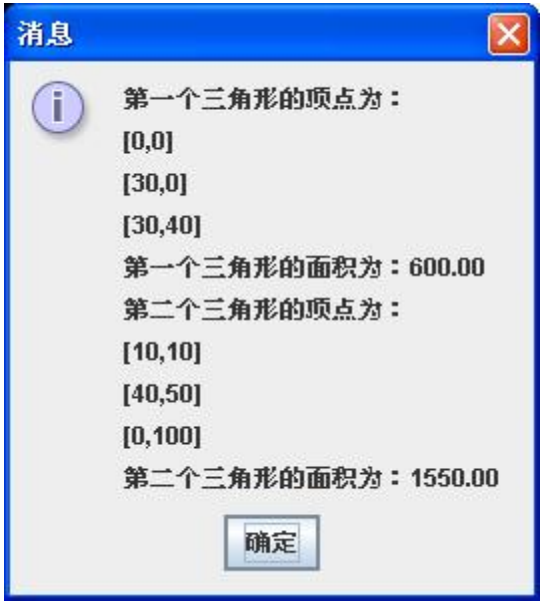
●在程序中使用对象实例包括引（使）用对象实例中的变量和调用对象实例中的方法，这都是通过**取成员运算符**“.”来实现的，其格式为：

```
objectReference.variable      //引用变量
objectReference.methodName([paramList]) //调用方法
```

●Java运行时系统通过自动**垃圾回收机制**周期性地清除无用对象，释放内存。

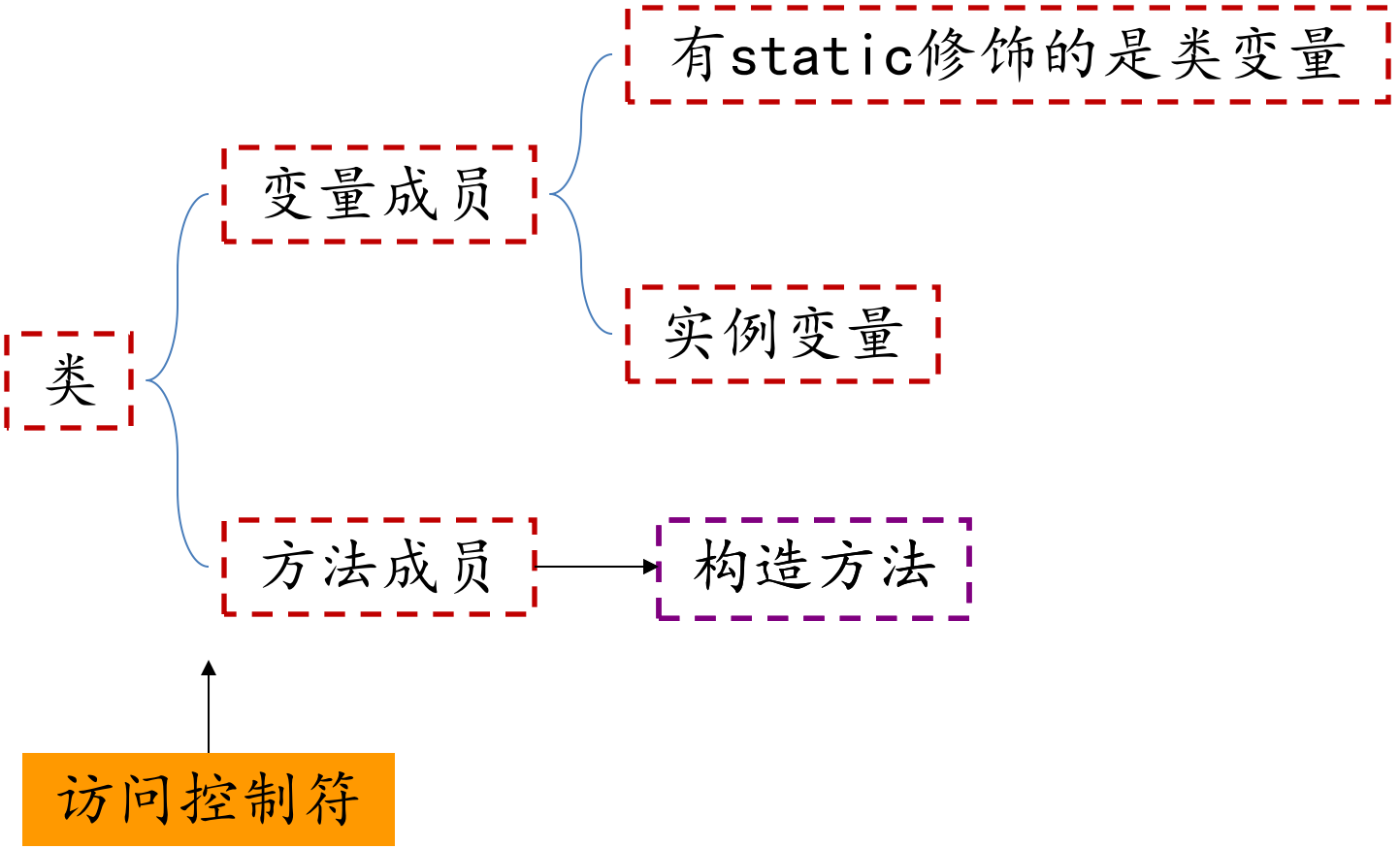
3.2 Java对象

```
import javax.swing.JOptionPane;
import java.text.DecimalFormat;
Import java.awt.Point;
Class Triangle{ …… }
public class TriangleTest {
    public static void main(String[] args)
    {          String output = "";
           Point a1,a2,a3;
           Point b1,b2,b3;
           a1 = new Point(0,0);                    //定义顶点
           a2 = new Point(30,0);
           a3 = new Point(30,40);
           b1 = new Point(10,10);
           b2 = new Point(40,50);
           b3 = new Point(0,100);
           Triangle t1,t2;                                  //定义三角形
           t1 = new Triangle(a1,a2,a3);
           t2 = new Triangle(b1,b2,b3);
           DecimalFormat twoDigits = new DecimalFormat("0.00");
           output += "第一个三角形的顶点为：\n"+t1.toString();
           output += "\n第一个三角形的面积为：
    "+twoDigits.format(t1.getTriangleArea());
           output += "\n"+"第二个三角形的顶点为：\n"+t2.toString();
           output += "\n第二个三角形的面积为：
    "+twoDigits.format(t2.getTriangleArea());
           JOptionPane.showMessageDialog(null,output);
           System.exit(0);
    }
```



3.1 Java类

◆变量成员与方法成员



3.2 Java对象

◆3.2.1对象的创建

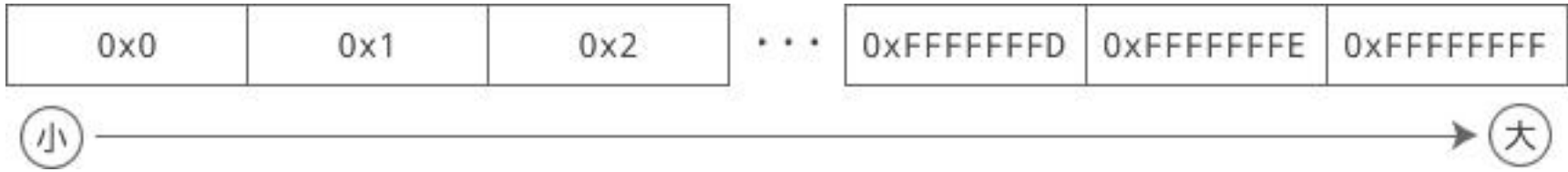
➢内存分配

```
Television tv;
Television tv = new Television();
```

●在对象通过new操作符创建以后，程序运行时，对象是怎么进行放置安排的呢？特别是内存是怎样分配的呢？

- 寄存器：这是最快的存储区，因为它位于不同于其他存储区域的地方--处理器内部。
- 栈：驻留于RAM（随机访问存储器）区域，但通过堆栈指针可以从处理器获得直接支持。
- 堆：一种通用的内存池（也位于RAM区），用于存放所有的Java对象。
- 静态域：存放Static定义的静态成员。
- 常量存储：常量值通常直接存放在代码内部，这样做是安全的，因为它们永远不会被改变。
- 非RAM存储：数据完全存活于程序之外。

Java的引用



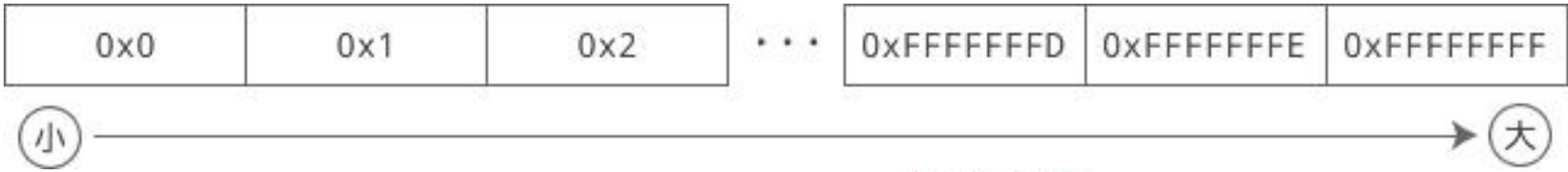
程序

数据

变量名和函数名为我们提供了方便，让我们在编写代码的过程中可以使用易于阅读和理解的英文字符串，不用直接面对二进制地址。

虽然变量名、函数名、字符串名和数组名在本质上是一样的，它们都是地址的助记符，但在编写代码的过程中，我们认为变量名表示的是数据本身，而函数名、字符串名和数组名表示的是代码块或数据块的首地址。

Java的引用




```c
#include <stdio.h>

int main(){
    int a = 100;
    char str[20] = "c.biancheng.net";
    printf("%#X, %#X\n", &a, str);
    return 0;
}
```

运行结果：

0X28FF3C, 0X28FF10

假设变量 a、b、c 在内存中的地址分别是 0X1000、0X2000、0X3000，那么加法运算c = a + b;将会被转换成类似下面的形式：

0X3000 = (0X1000) + (0X2000);

变量名和函数名为我们提供了方便，让我们在编写代码的过程中可以使用易于阅读和理解的英文字符串，不用直接面对二进制地址。

虽然变量名、函数名、字符串名和数组名在本质上是一样的，它们都是地址的助记符，但在编写代码的过程中，我们认为变量名表示的是数据本身，而函数名、字符串名和数组名表示的是代码块或数据块的首地址。

Java的引用

C++

引用：就是某一变量（目标）的一个别名，对引用的操作与对变量直接操作完全一样。

引用的声明方法：类型标识符 &引用名=目标变量名；

（1）&在此不是求地址运算符，而是起标识作用。

（2）类型标识符是指目标变量的类型。

（3）声明引用时，必须同时对其进行初始化。

（4）引用声明完毕后，相当于目标变量有两个名称即该目标原名称和引用名，且不能再把该引用名作为其他变量名的别名。

（5）声明一个引用，不是新定义了一个变量，它只表示该引用名是目标变量名的一个别名，它本身不是一种数据类型，因此引用本身不占存储单元，系统也不给引用分配存储单元。故：对引用求地址，就是对目标变量求地址。

（6）不能建立数组的引用。因为数组是一个由若干个元素所组成的集合，所以无法建立一个数组的别名。

Java的引用

C++

1.指针和引用的定义和性质区别：

(1)指针：指针是一个变量，只不过这个变量存储的是一个地址，指向内存的一个存储单元；而引用跟原来的变量实质上是同一个东西，只不过是原变量的一个别名而已。

(2)可以有const指针，但是没有const引用；

(3)指针可以有多级，但是引用只能是一级（int **p；合法 而 int &&a是不合法的）

(4)指针的值可以为空，但是引用的值不能为NULL，并且引用在定义的时候必须初始化；

(5)指针的值在初始化后可以改变，即指向其它的存储单元，而引用在进行初始化后就不会再改变了。

(6)"sizeof引用"得到的是所指向的变量(对象)的大小，而"sizeof指针"得到的是指针本身的大小；

(7)指针和引用的自增(++)运算意义不一样；

C++

Java的引用

```
void swap(int *a,int *b)
{
        int temp=*a;
        *a=*b;
        *b=temp;
}

int main(void)
{
        int a=1,b=2;
        swap(&a,&b);
        cout<<a<<" "<<b<<endl;
        system("pause");
        return 0;
}
```

结果为2 1

```
void test(int *p)
{
        int a=1;
        p=&a;
        cout<<p<<" "<<*p<<endl;
}

int main(void)
{
    int *p=NULL;
    test(p);
    if(p==NULL)
    cout<<"指针p为NULL"<<endl;
    system("pause");
    return 0;
}
```

0x22ff44 1
指针p为NULL

```
void test(int &a)
{
        cout<<&a<<" "<<a<<endl;
}

int main(void)
{
    int a=1;
    cout<<&a<<" "<<a<<endl;
    test(a);
    system("pause");
    return 0;
}
```

0x22ff44 1
0x22ff44 1

提高代码执行效率和增强代码质量的一个很好的办法。

Java的引用

```
package test;

public class TestReference {
    public void test(Reference p){
        Reference a = new Reference();
        p = a;
        System.out.println(p);
        System.out.println(a);
    }
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Reference p = null;
        TestReference t = new TestReference();
        t.test(p);
        if(p==null)
            System.out.println("p为null");
    }
}
class Reference {
}
```

C++ 指针 引用

Java 引用

test.Reference@1bb60c3
test.Reference@1bb60c3
p为null

3.2 Java对象

◆对象初始化Initialization of object

●解决对象初始化问题是保障程序安全的基本之一。构造方法是解决初始化问题的一种方法。

●**局部变量**，Java以编译时错误的形式来贯彻这种保证。

```
void f() {
    int i;
    i++; // Error, i is not initialized}
```

●然而，若**变量成员**是基本类型，编辑器会给定初始值。

3.2 Java对象

◆对象初始化

```
public class InitialValues {
    boolean t;
    char c;
    byte b;
    short s;
    int i;
    long l;
    float f;
    double d;
    InitialValues reference;
    void printInitialValues() {
        System.out.println(
                "Data type Inital value\n" +
                "boolean " + t + "\n" +
                "char " + c + "\n" +
                "byte " + b + "\n" +
                "short " + s + "\n" +
                "int " + i + "\n" +
                "long " + l + "\n" +
                "float " + f + "\n" +
                "double " + d +
                “reference ” + reference);
}
```

```
public static void main(String[] args) {
    InitialValues iv = new InitialValues();
    iv. printInitialValues();
    }
}
/*output:
Data type Inital value
boolean false
char
byte 0
short 0
int 0
long 0
float 0.0
double 0.0
reference null
*/
```

3.2 Java对象

◆对象初始化

- 默认值
- 声明时设定初值
- 通过方法返回值赋初值
- 构造方法初始化

基本类型	默认值
boolean	flase
char	‘\u0000’(null)
byte	(byte)0
short	(short)0
int	0
long	0L
float	0.0f
double	0.0d

◆成员初始化次序

●在类的内部，变量定义的先后顺序决定了初始化的顺序。

●即使变量定义散布于方法定义之间，它们仍旧会在任何方法被调用之前得到初始化。

```
class Window {
    Window(int marker) {
      System.out.println("Window("+ marker +") ");
    }}
class House {
    Window w1 = new Window(1);
    House() {
        System.out.println("House()");
        W3 = new Window(33);}
    Window w2 = new Window(2);
    void f() {
        System.out.println("f()");}
    Window w3 = new Window(3);
}
```

```
public class OrderInitialization{
    public static void main(String[] args) {
            House h = new House();
            h.f();
    }
}
/* Output:
Window(1)
Window(2)
Window(3)
House()
Window(33)
f()
*/
```

3.2 Java对象

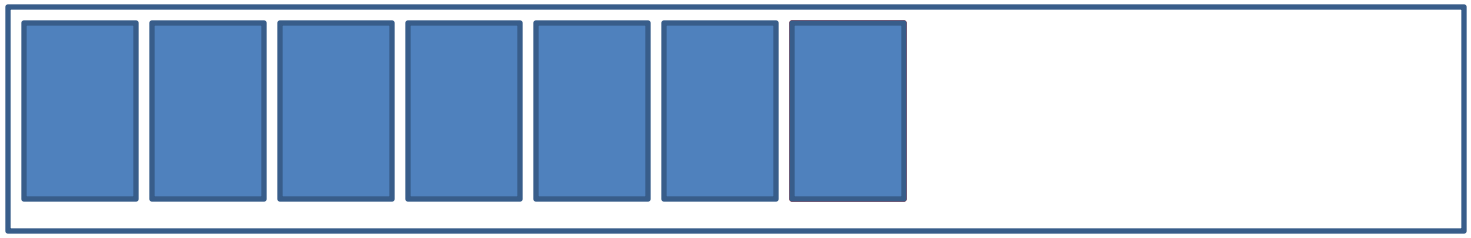
◆垃圾回收机制

- 垃圾回收机制GC(Garbage Collection)是Java语言的核心技术之一。
- 不需要程序员手动释放对象占用的内存资源。
- GC通过确定对象是否被活动对象引用来确定是否收集该对象。
- 提高堆上内存分配的速度。
- 垃圾收集的一个潜在的缺点是它的开销影响程序性能。
- 垃圾收集的算法。

3.2 Java对象

◆垃圾回收机制

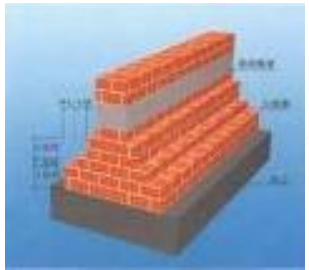
内存

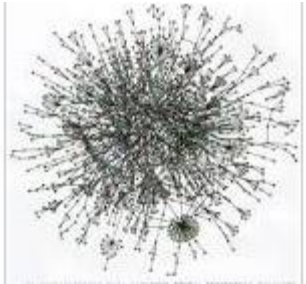


提高内存分配速度

3.3Package/Class Library

- 3.3.1 命名空间
- 3.3.2 package语句
- 3.3.3 import语句





3.3包：库单元

◆命名空间Namespace

图书系统、自己的文件、域名

www.jlu.edu.cn
csw.jlu.edu.cn
ccst.jlu.edu.cn

3.3包：库单元

- 命名空间




3.3包：库单元

◆命名空间

➢Java引入了包的机制，以提供类的多重命名空间，同时也负责类名空间管理。把因不同目的和不同工作而开发的类放在不同的包中，即使在出现相同的类名的时候，也可以很好地管理。包可以有一定的层次，对应着外存上的目录结构。

➢为了区别于各种平台Java中采用了“.”来分隔目录，与域名地址不同的是，Java将主类从左侧写起。

3.3包：库单元

```
//: net/mindview/simple/Vector.java
// Creating a package.
package net.mindview.simple;

public class Vector {
  public Vector() {
    System.out.println("net.mindview.simple.Vector");
  }
} ///:~
```

```
//: net/mindview/simple/List.java
// Creating a package.
package net.mindview.simple;

public class List {
  public List() {
    System.out.println("net.mindview.simple.List");
  }
} ///:~
```

package net.mindview.simple
\net\mindview\simple\
Vector.class C:\DOC\JavaT\net\mindview\simple\Vector.class
List.class C:\DOC\JavaT\net\mindview\simple\List.class

Java interpreter → *CLASSPATH* → directory → .class file

```
CLASSPATH=.;D:\JAVA\LIB;C:\DOC\JavaT
```

3.3包：库单元

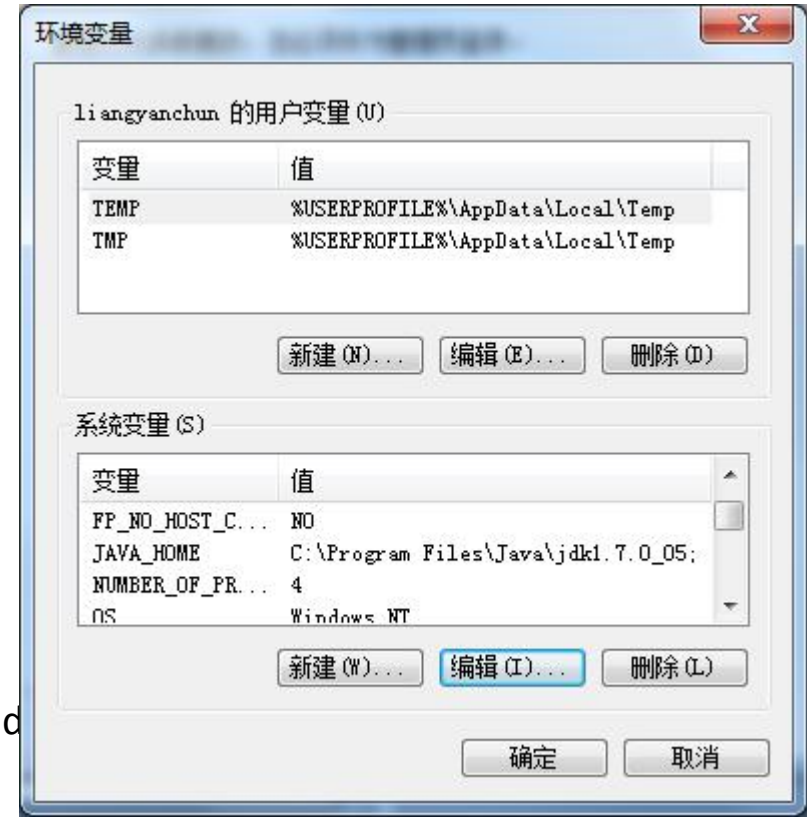
```
2014-02-11  13:42    <DIR>          Public
               0 个文件                0 字节
               4 个目录 29,719,154,688 可用字节

C:\Users>java
用法: java [-options] class [args...]
           (执行类)
   或  java [-options] -jar jarfile [args...]
           (执行 jar 文件)
其中选项包括:
    -d32          使用 32 位数据模型 (如果可用)
    -d64          使用 64 位数据模型 (如果可用)
    -client       选择 "client" VM
    -server       选择 "server" VM
    -hotspot      是 "client" VM 的同义词 [已过时]
                  默认 VM 是 client.

    -cp <目录和 zip/jar 文件的类搜索路径>
    -classpath <目录和 zip/jar 文件的类搜索路径>
                  用 ; 分隔的目录, JAR 档案
                  和 ZIP 档案列表, 用于搜索类文件。
```

```
C:\Users>play
'play' 不是内部或外部命令，也不是可运行的程序
或批处理文件。

C:\Users>
```



Java interpreter → *CLASSPATH* → d

```
CLASSPATH=.;D:\JAVA\LIB;C:\DOC\JavaT
```

3.3包：库单元

◆命名空间

- Java中提供了大量的包（类库）

  java.util、java.lang、java.swing

- 也可以建立自己的包

  maindirectory.function.basicfunction

3.3包：库单元

◆package

●package语句放在Java源程序文件的第一行，指明该文件中定义的类存放到哪个包中。程序中可以没有package语句，此时类将存放到默认包中。

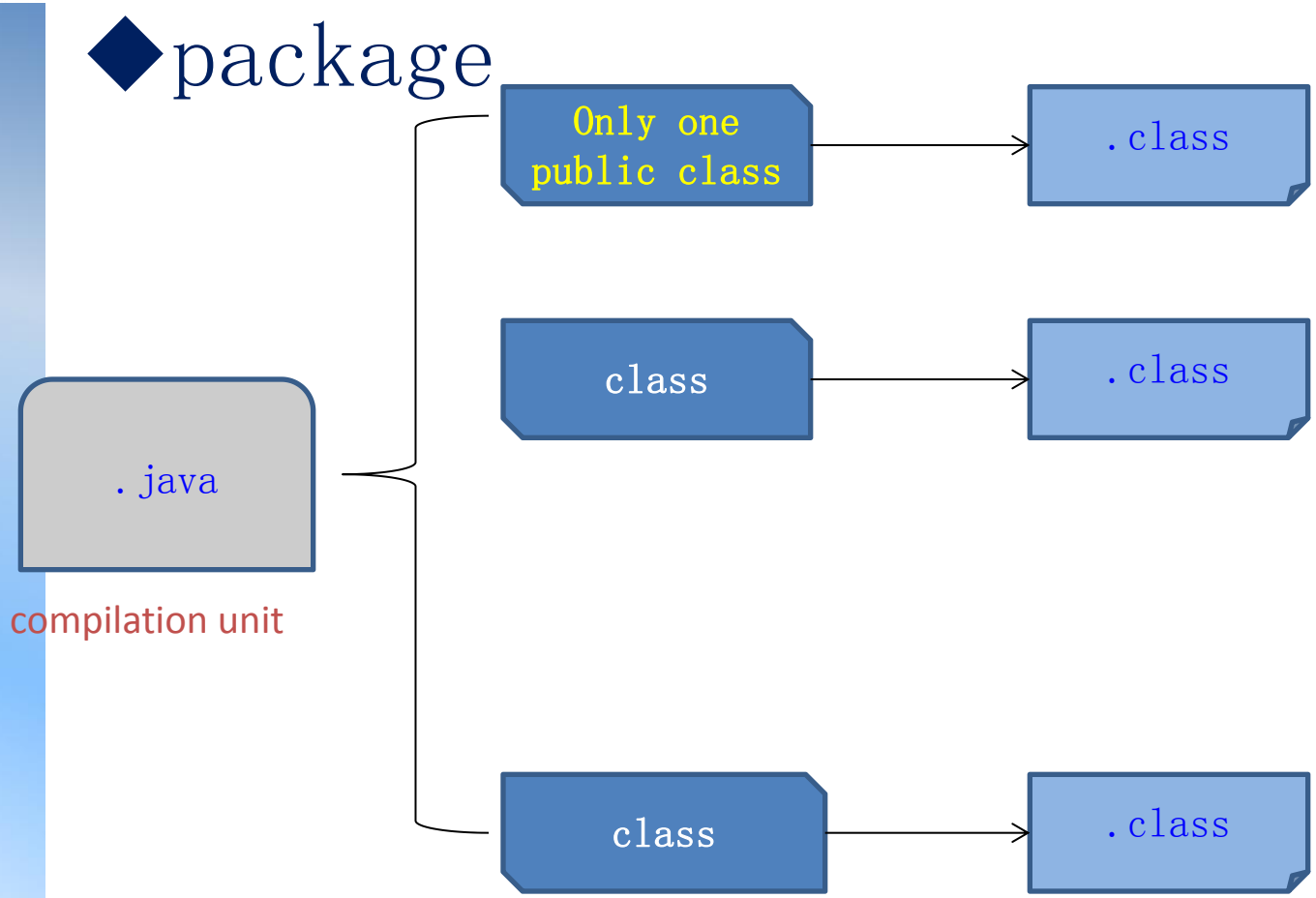
package pkg1.[pkg2.[pkg3….]];

package access.mypackage;

●显然package语句最多只能有一条。

●描述包的字符都用小写。

3.3包：库单元



- public class: 在每个编译单元中有且仅有一个公共类。
- public class name: 类的名字与文件的名字相同，包括大小写，后缀名为.java。
- additional classes: 主类的支持类。

3.3包：库单元

◆package

```
package access.mypackage;

public class MyClass {
//…
}
```

●MyClass是命名在access.mypackage包中的，要想使用这个类，需要用import关键字来使用access中的名称。

3.3包：库单元

import

- import语句的作用是引入所需的类，以供在程序中使用类库中现有的类。

  import pkg1[.pkg2…].classname|*；

- import语句在程序中放在package语句之后，类定义之前。一个Java源程序中可以有多条import语句。

```
public class FullQualification {
    public static void main(String[] args) {
            java.util.ArrayList list = new
            java.util.ArrayList();
    }
}
```

```
import java.util.ArrayList;
    public class SignleImport {
        public static void main(String[] args) {
            ArrayList list = new ArrayList();
        }
}
```

3.3包：库单元

import

- 要用java.util中的其他类

```
import java.util.*
```

- 避免冲突

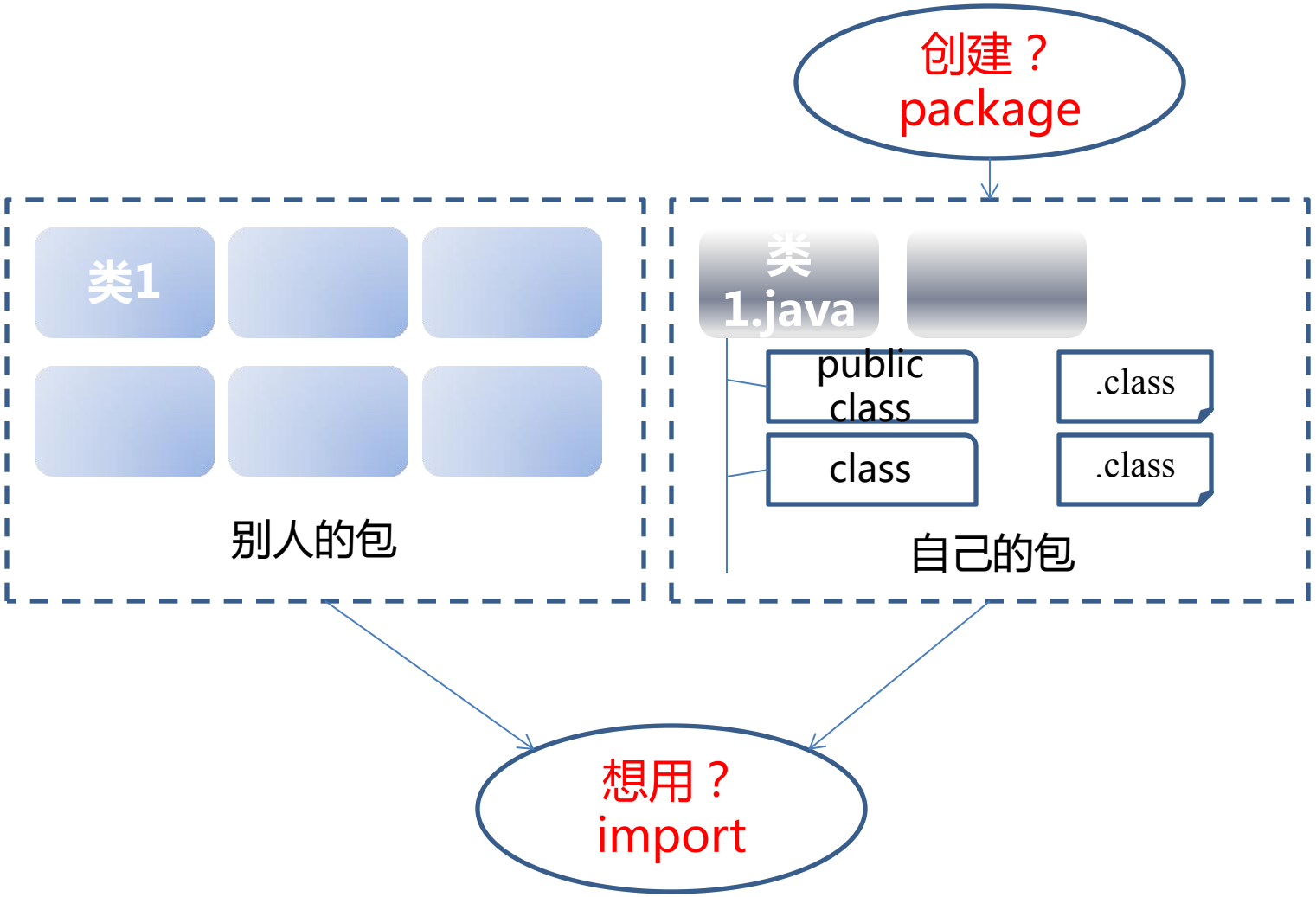
```
import java.util.*;

import access.mypackage.Vector;

Vector v = new Vector(); //error

java.util.Vector v = new java.util.Vector();
```

3.3包：库单元


创建？
package
类1
别人的包
类
1.java
public
class
class
.class
.class
自己的包
想用？
import

3.4 Java标准类库

- 3.4.1 Java.lang包和Java.util包
- 3.4.2 Object类
- 3.4.3 System类
- 3.4.4 Math类
- 3.4.5 String类、StringBuffer类、StringBuilder类

3.4 Java标准类库

◆面向对象技术是提高软件可重用性最有效的途径之一，其中类的引用和类的继承机制为提高程序的可重用性提供了强有力的支持。

◆Java 有一系列功能强大的类库，功能相关的可重用类被组织成包。可重用类的继承层次和包的组织呈树型结构。我们在进行编程时要首先考虑如何充分利用已有可重用类来构造自己的新类。因此，了解和掌握Java可重用类是学习Java 程序设计的重要方面。

3.4 Java标准类库

◆常用库

●java.lang包：提供了Java程序设计中最基础的类。如，有常用的String 类、Object 类等。

●java.util包：包括许多具有特定功能的类，如Arrays，Date，Calendar和Stack 等。

●java.io包：主要包含与输入输出相关的类，这些类提供了对不同的输入和输出设备读写数据的支持。

●javax.swing包：提供了创建图形用户界面元素的类。该包中包含定义窗口、对话框、按钮、复选框、列表、菜单、滚动条及文本域的类。

●java.net包：包含与网络操作相关的类，如URL、InetAddress 和Socket 等。

3.4 Java标准类库

◆java.lang包

●java.lang包由解释程序自动加载，不需在程序中显式地使用语句import java.lang.*。

●提供常用的类、接口、一般异常、系统等编程语言的核心内容。

基本数据类型、基本数学函数、字符串处理、线程、异常处理类等

3.4 Java标准类库

◆ java.lang包

Boolean	Long	StrictMath
Byte	Math	String
Character	Number	StringBuffer
Class	Object	System
ClassLoader	Package	Thread
Compiler	Process	ThreadGroup
Double	Runtime	ThreadLocal
Float	RuntimePermission	Throwable
InheritableThreadLocal	SecurityManager	Void
Integer	Short	

Appendable	能够被添加char序列和值的对象。
CharSequence	是char值的一个可读序列。
Cloneable	此类实现了Cloneable接口，以指示 Object.clone()方法可以合法地对该类实例进行按字段复制。
Comparable	此接口强行对实现它的每个类的对象进行整体排序。
Iterable	实现这个接口允许对象成为“foreach”语句的目标。
Readable	Readable是一个字符源。
Runnable	Runnable接口应该由那些打算通过某一线程执行其实例的类来实现。
Thread.UncaughtExceptionHandler	当Thread因未捕获的异常而突然终止时，调用处理程序的接口。

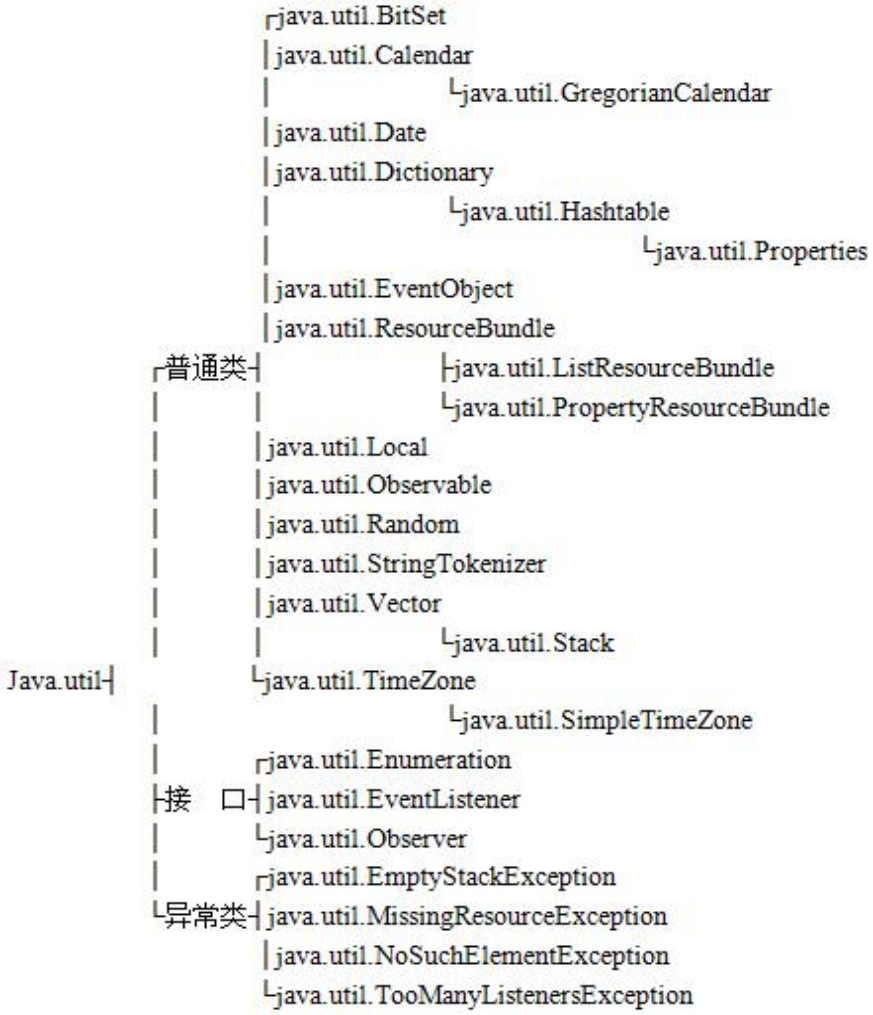
3.4 Java标准类库

◆java.util包

- java.util包是Java实用工具类库。
- 提供了一些实用的方法和数据结构。
- Java提供日期(Data)类、日历(Calendar)类来产生和获取日期及时间，提供随机数(Random)类产生各种类型的随机数，还提供了堆栈(Stack)、向量(Vector)、位集合(Bitset)以及哈希表(Hashtable)等类来表示相应的数据结构。

3.4 Java标准类库

◆java.util包



3.4 Java标准类库

◆Object类

- java.lang.Object
- Object类是所有Java类的基类,它处于Java开发环境的类层次树的根部。
- 如果一个类在定义的时候没有包含extends关键字，编译器会将其作为Object类的直接子类。
- Object类定义了所有对象的最基本的一些状态和行为，它提供的方法会被Java中的每个类所继承。
- 可以使用类型为Object的变量引用任意类型的对象。

```
Object obj = new MyObject();
MyObject x = (MyObject)obj;
```

3.4 Java标准类库

◆Object类

```
public class java.lang.Object{
   public Object();                                    //构造方法
   protected Object clone();                           //建立当前对象的拷贝
   public boolean equals(Object obj);                  //比较对象
   protected void finalize();                          //释放资源
   public final Class getClass();                      //求对象对应的类
   public int hashCode();                              //求hash码值
   public final void notify();                         //唤醒当前线程
   public final void notifyAll();                      //唤醒所有线程
   public String toString();                           //返回当前对象的字符串
   public final void wait();                           //使线程等待
   public final void wait(long timeout);
   public final void wait(long timeout, int nanos);
// 其中timeout为最长等待时间，单位为毫秒；nanos为附加时间，单位为纳秒，
// 取值范围为0～999999。
}
```

3.4 Java标准类库

◆Object类

```
import java.awt.*;
public class ObjectExam {
        public static void main(String[] args) {
                Integer a=new Integer(1);
                Integer b=new Integer(1);
                Rectangle c=new Rectangle(20,5);
                System.out.println(a.equals(b));
                System.out.println("The Object class is:"+a.getClass());
                System.out.println(c.toString());
        }
}
/*output:
True
The Object class is:class java.lang.Integer
java.awt.Rectangle[x=0,y=0,width=20,height=5]
*/
```

3.4 Java标准类库

◆System类

- Java.lang.System
- System包含了一些我们常用的方法与成员变量。
- System不能被实例化， 所有的方法都可以直接引用。
- System类提供了标准输入（in）输出（out）和错误（error）流。

```
public static void main(String[] args) {
    System.out.println("Hello System.out!");
    System.err.println("Hello System.err!");
}
/*output:
Hello System.out!
Hello System.err!
*/
```

3.4 Java标准类库

◆System类

- System还提供了系统属性的获取和设置功能。getProPerty方法可以用来获取系统的属性。

```
System.getProperty（”OS.name”）；//可获得当前系统的操作系统名字。
System.getProperty（”Java.version”）；//可获得Java版本号。
```

- System还提供了setProperties方法用来设置或修改系统的属性；
- currentTimeMillis方法来获取当前时间；
- exit方法使程序退出系统；
- arraycopy方式来拷贝数组；
- 等等。

3.4 Java标准类库

◆Math类

- java.lang.Math
- 包含完成基本数学函数所需的方法。
- 静态方法，不用声明具体的实例。
- 主要包括三角函数方法、指数方法、服务方法、随机数方法。
- Math中还声明了两个double型的常量，PI和E可以在任何程序中用Math.PI和Math.E。

3.4 Java标准类库

◆Math类

```
Math.sin(0); // 0.0
Math.sin(Math.PI / 6); // 0.5
Math.cos(Math.PI / 6); // 0.866
Math.toDegrees(Math.PI / 2); // 90.0
Math.toRadians(30); // /6

Math.exp(1);//2.71828 返回e的1次方
Math.pow(2,3);//8 返回2的3次方
Math.sqrt(4);//2 返回4的平方根

Math.random();//随机生成[0.0,1.0]之间的随机数
a+Math.random()*b;//可以生成[a,a+b]之间的随机数
(int)(Math.random()*a);//可以生成[0,a]之间的随机整数


public static double ceil(double x)//是向上取整，
public static double floor(double x)//是向下取整，
public static double min(double x, double y)//是求小值，
public static double max(double x, double y)//是求大值
public static double abs(double x)//是求绝对值
```

3.4 Java标准类库

◆String类

- java.lang.String
- 字符串（string）与人类语言很接近，也是计算机程序设计中最常见的行为。
- 多个构造方法：

```
String(byte[] bytes);
String(byte[] bytes,int offset, int length);
String(byte[] bytes, int offset, int length, int count);
String(char[] value);
String(char[] value, int offset, int count);
String(String original);
String(StringBuffer buffer);
```

```
String s1 = “hello Java”;
String s2 = new String(“welcome to String”);
char c[ ]= {'s','u','n',' ','j','a','v','a'};
String s3 = new String(c);
String s4 = new String(c,4,4);
```

3.4 Java标准类库

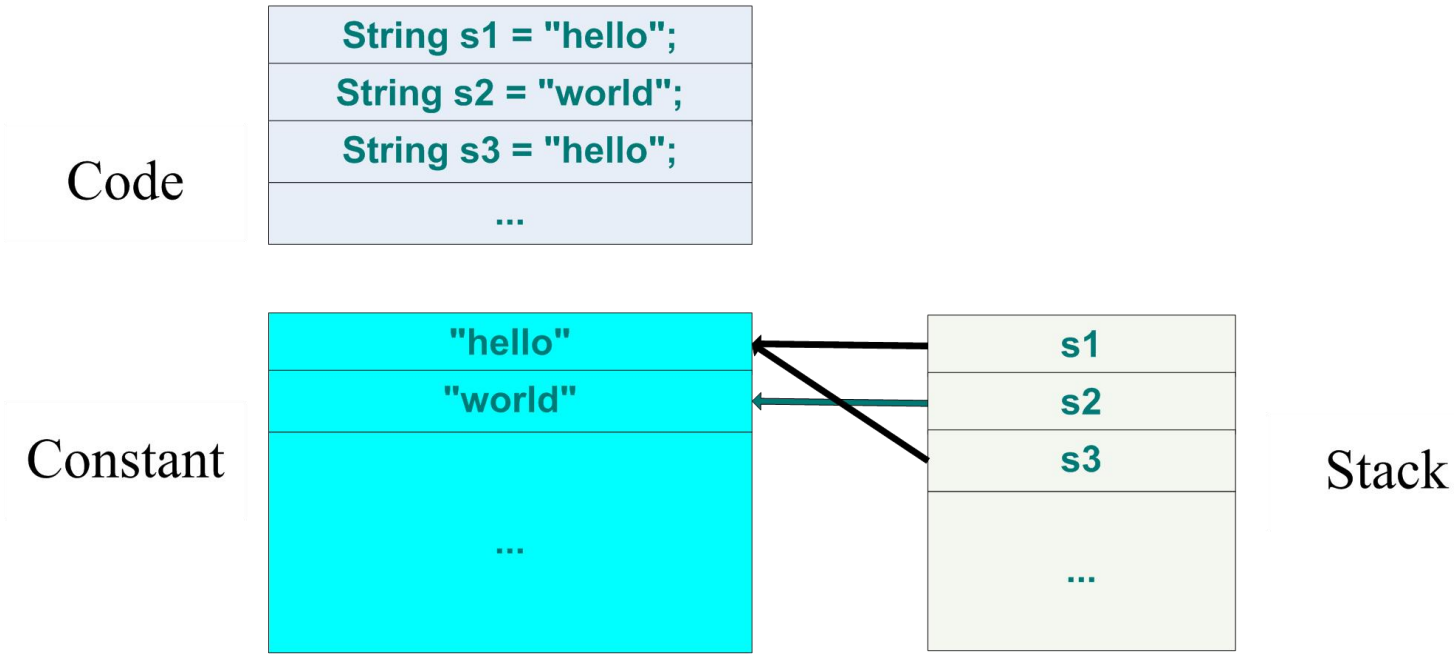
◆String类

- 类String用来处理不变的字符串常量。

```
public class Test {
    public static void main (String[ ] args) {
        String s1 = "hello";
        String s2 = “world”;
        String s3 = "hello";
        System.out.println(s1 == s3); //true
        s1 = new String ("hello");
        s2 = new String ("hello") ;
        System.out.println(s1 == s2); //false
        System.out.println(s1.equals(s2)); //true
    }
}
```

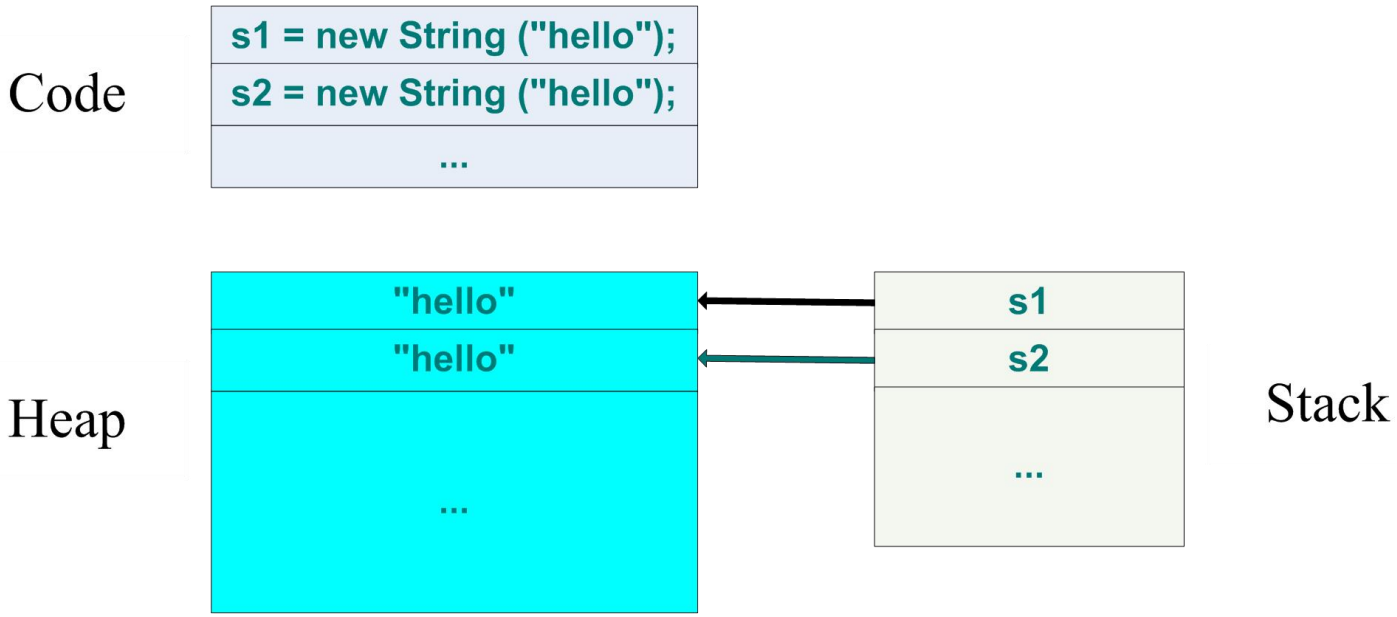
3.4 Java标准类库

◆String类



3.4 Java标准类库

◆String类



http://blog.csdn.net/langhong8/article/details/50938041
https://blog.csdn.net/qq_27093465/article/details/52033327

3.4 Java标准类库

◆String类

```
1 String s1 = "Hello";
 2 String s2 = "Hello";
 3 String s3 = "Hel" + "lo";
 4 String s4 = "Hel" + new String("lo");
 5 String s5 = new String("Hello");
 6 String s6 = s5.intern();
 7 String s7 = "H";
 8 String s8 = "ello";
 9 String s9 = s7 + s8;
10
11 System.out.println(s1 == s2);  // true
12 System.out.println(s1 == s3);  // true
13 System.out.println(s1 == s4);  // false
14 System.out.println(s1 == s9);  // false
15 System.out.println(s4 == s5);  // false
16 System.out.println(s1 == s6);  // true
```

http://blog.csdn.net/langhong8/article/details/50938041
https://blog.csdn.net/qq_27093465/article/details/52033327

3.4 Java标准类库

◆String类

- 类String提供了大量的方法，使得字符串的处理更加方便。

public char charAt(int index) //返回指定索引处的char值

public int length()//返回此字符串的长度

public int indexOf(String str) //返回指定字符在第一次出现处的索引

public boolean equalsIgnoreCase(String another) //忽略大小写的相等

public String replace(char oldChar,char newChar) //替换所有匹配字符

3.4 Java标准类库

◆String类

```
public class Test {
    public static void main (String[] args) {
            String s1 = "sun java", s2 = "Sun Java";
            System.out.println(s1.charAt(1)) ;//u
            System.out.println(s2.length()) ;//8
            System. out. println (s1. indexOf ("java") ) ; // 4
            System. out. println (s1. indexOf ("Java") ) ; // -1
            System.out.println(s1.equals(s2)) ;//false
            System.out.println(s1.equalsIgnoreCase(s2)); //true
            String s = “He is a student. He is studying java now.";
            String sr = s.replace("He","She") ;
            System.out.println(sr) ; // She is a student. She is studying java now.
  }}
```

3.4 Java标准类库

◆String类

●提供的方法：

public boolean startsWith(String prefix) //是否开始于一个字符串

public boolean endsWith(String suffix) //是否结束于一个字符串

public String toUpperCase() //转化成大写

public String toLowerCase() //转化成小写

public String substring(int beginIndex) //开始于begin位置的子字符串

public String substring(int beginIndex,int endIndex) //开始于begin位置，结束于end位置的子字符串

public String trim()//去掉字符串两端的空格

3.4 Java标准类库

◆String类

```
public class Test {
    public static void main (String[] args) {
            String s = "Welcome to Java World!";
            String s1 =" sun java ";
            System. out .println (s.startsWith("Welcome")); //true
            System. out .println (s.endsWith("World")); //false
            String sL = s.toLowerCase();
            String sU = s.toUpperCase();
            System.out.println(sL) ;//welcome to java world!
            System.out.println(sU) ;//WELCOME TO JAVA WORLD!
            String subS = s.substring(11);
            System.out.println(subS) ;//Java World!
            String sp = s1.trim();
            System.out.println(sp) ;//sun java
    }
}
```

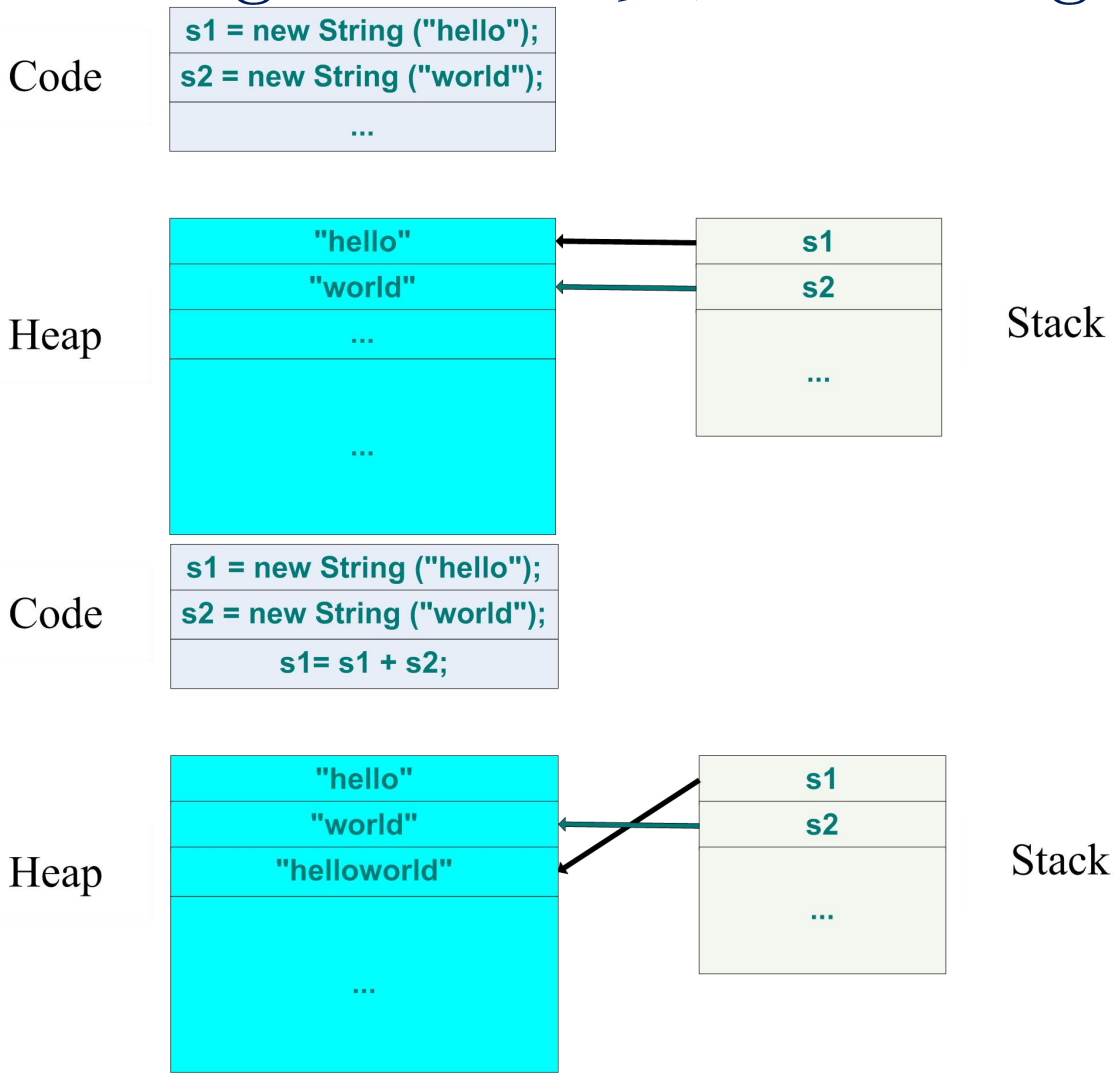
3.4 Java标准类库

◆StringBuilder类、StringBuffer类

- Java.lang.StringBuilder
- Java.lang.StringBuffer
- 比String类更灵活，可以给一个StringBuilder或StringBuffer中删除、插入或追加新的内容。
- String对象一旦创建，它的值就确定了。

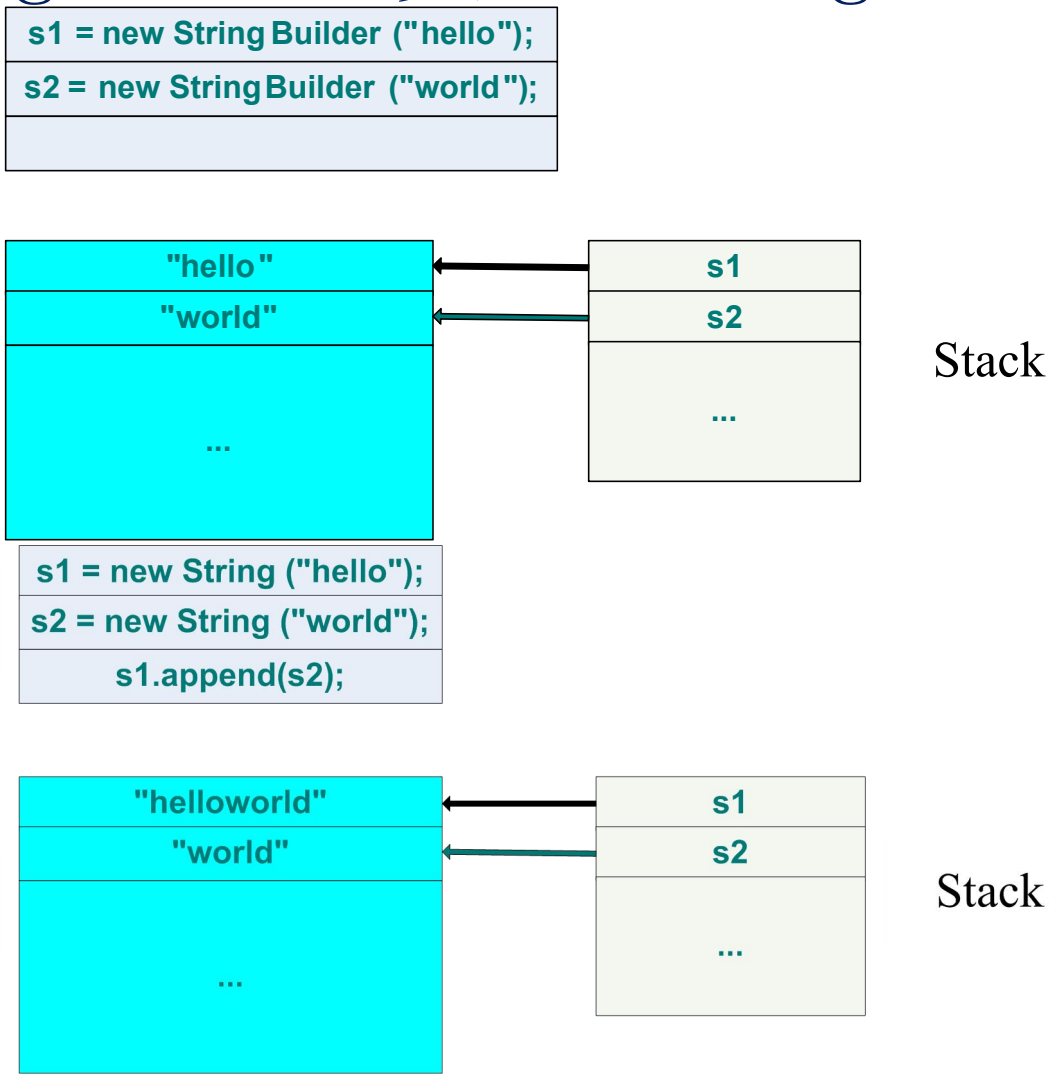
3.4 Java标准类库

◆StringBuilder类、StringBuffer类



3.4 Java标准类库

◆StringBuilder类、StringBuffer类



3.4 Java标准类库

◆StringBuilder类、StringBuffer类

●String和StringBuilder的区别，对于经常修改的字符串，StringBuilder的运算时间远远好于String，感兴趣的同学可以做一下下面的例子：

```
public class AppendStringTest {
    public static void main(String[] args) {
        String text = "";
        long beginTime = System.currentTimeMillis();
        for(int i = 0; i < 10000; i++)
            text = text + i;
        long endTime = System.currentTimeMillis();
        System.out.println( "Run Time：  " + (endTime - beginTime));
        StringBuilder builder = new StringBuilder( " ");
        beginTime = System.currentTimeMillis();
        for(int i = 0; i < 10000; i++)
            builder.append(String.valueOf(i));
        endTime = System.currentTimeMillis();
        System.out.println( "Run Time：  " + (endTime - beginTime));
    } }
```

3.4 Java标准类库

◆StringBuilder类、StringBuffer类

- 在java中text = text +i是通过下面的方式实现的：

```
StringBuilder builder = new StringBuilder(text);
builder.append(String.valueOf(i));
text = builder.tostring();
```

- StringBuilder和StringBuffer功能上是相同的，StringBuffer是线程安全的，StringBuilder类被设计用作StringBuffer 的一个简易替换。

- 建议优先采用StringBuilder类，因为在大多数实现中，它比StringBuffer要快。

3.4 Java标准类库

◆StringBuilder类、StringBuffer类

●StringBuilder中常用的方法：

public StringBuilder append (...) //追加参数（多种类型）中的内容到字符串中

public StringBuilder insert (...) //将任意参数的字符串形式插入到原有字符串指定的位置

public StringBuilder delete (int start, int end) //删除从start开始到end-1为止的一段字符序列

public StringBuilder reverse (StringBuilder str) //将字符序列逆序

3.4 Java标准类库

◆StringBuilder类、StringBuffer类

```
public class Test {
        public static void main(String[] args) {
                String s = "Mircosoft";
                char [] a = {'a','b','c'} ;
                StringBuffer sbl = new StringBuffer(s);
                sbl.append ('/').append("IBM").append('/').append ("Sun");
                System.out.println(sbl) ;
                StringBuffer sb2 = new StringBuffer("nu");
                for(int i = 0;i<=9;i++) {sb2.append(i) ;}
                System.out.println(sb2) ;
                sb2.delete(8,sb2.length()).insert(0,a);
                System.out.println(sb2) ;
                System.out.println(sb2.reverse()) ;
        }
}
/*output:
Microsoft/IBM/Sun
nu0123456789
abcnu012345
543210uncba
*/
```

内容提要

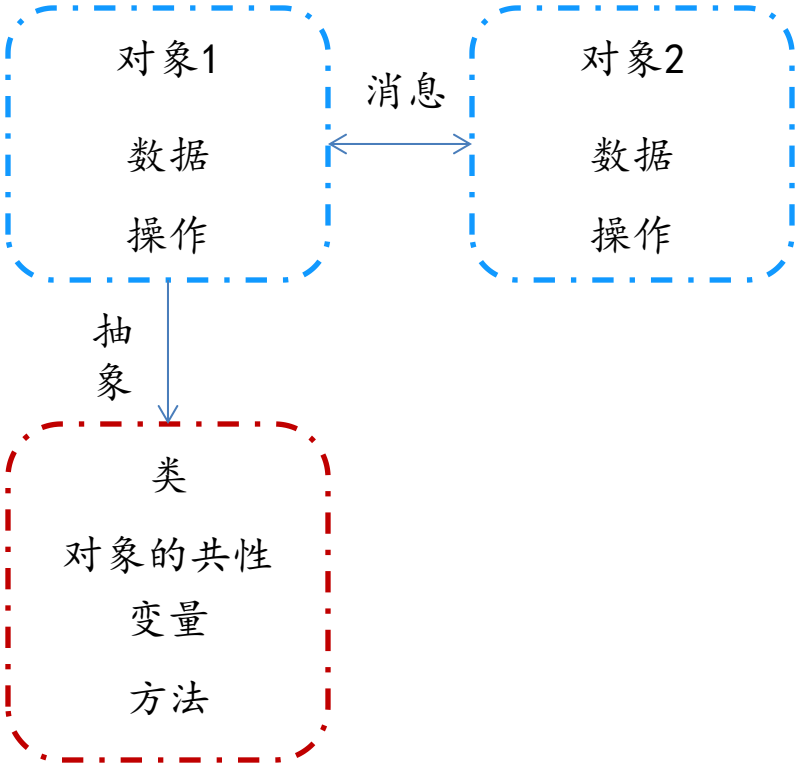
类与对象

Java 基本语法

Java 语言背景、历史、特点

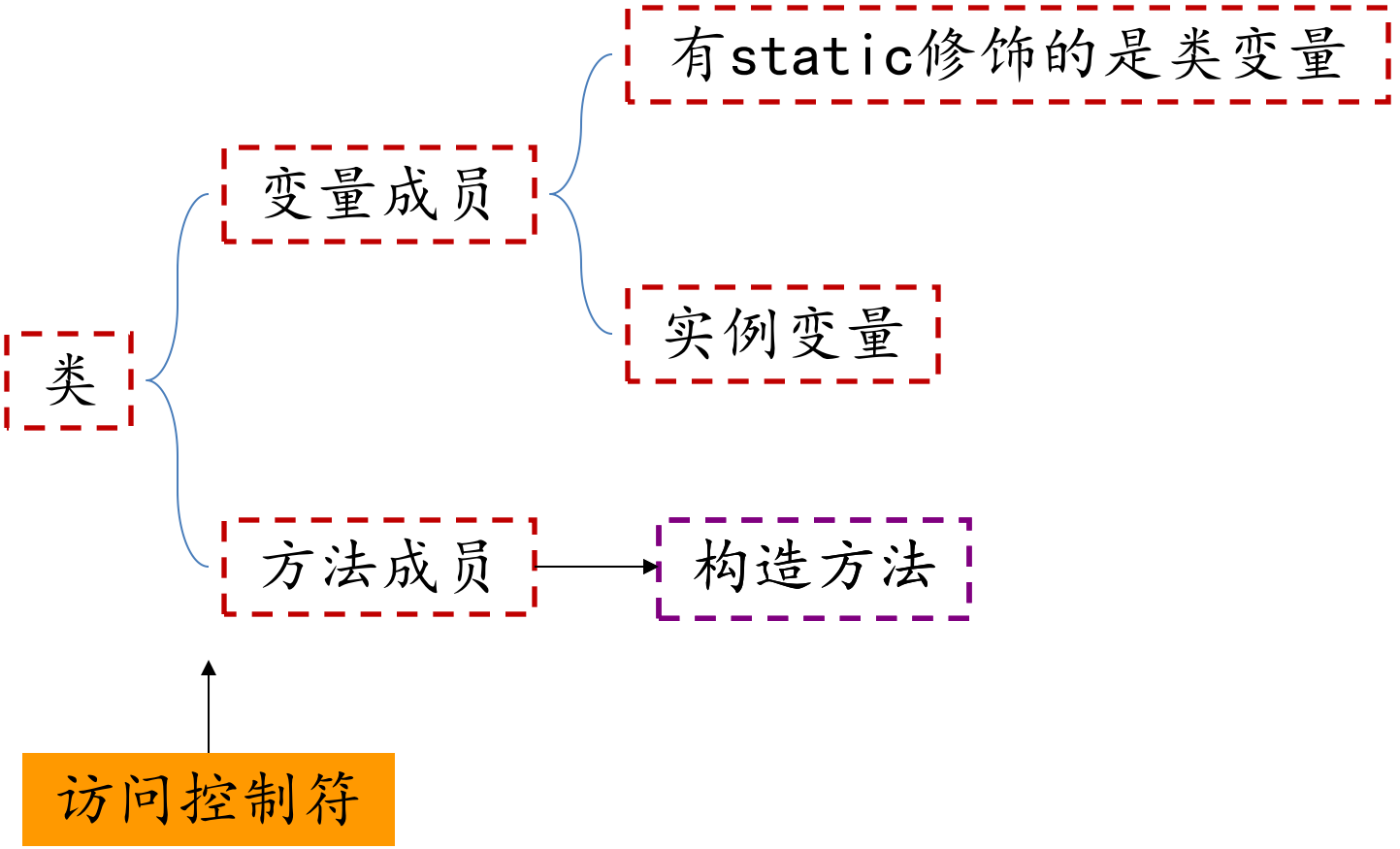
面向对象的程序设计

◆封装：对象、类和消息



3.1 Java类

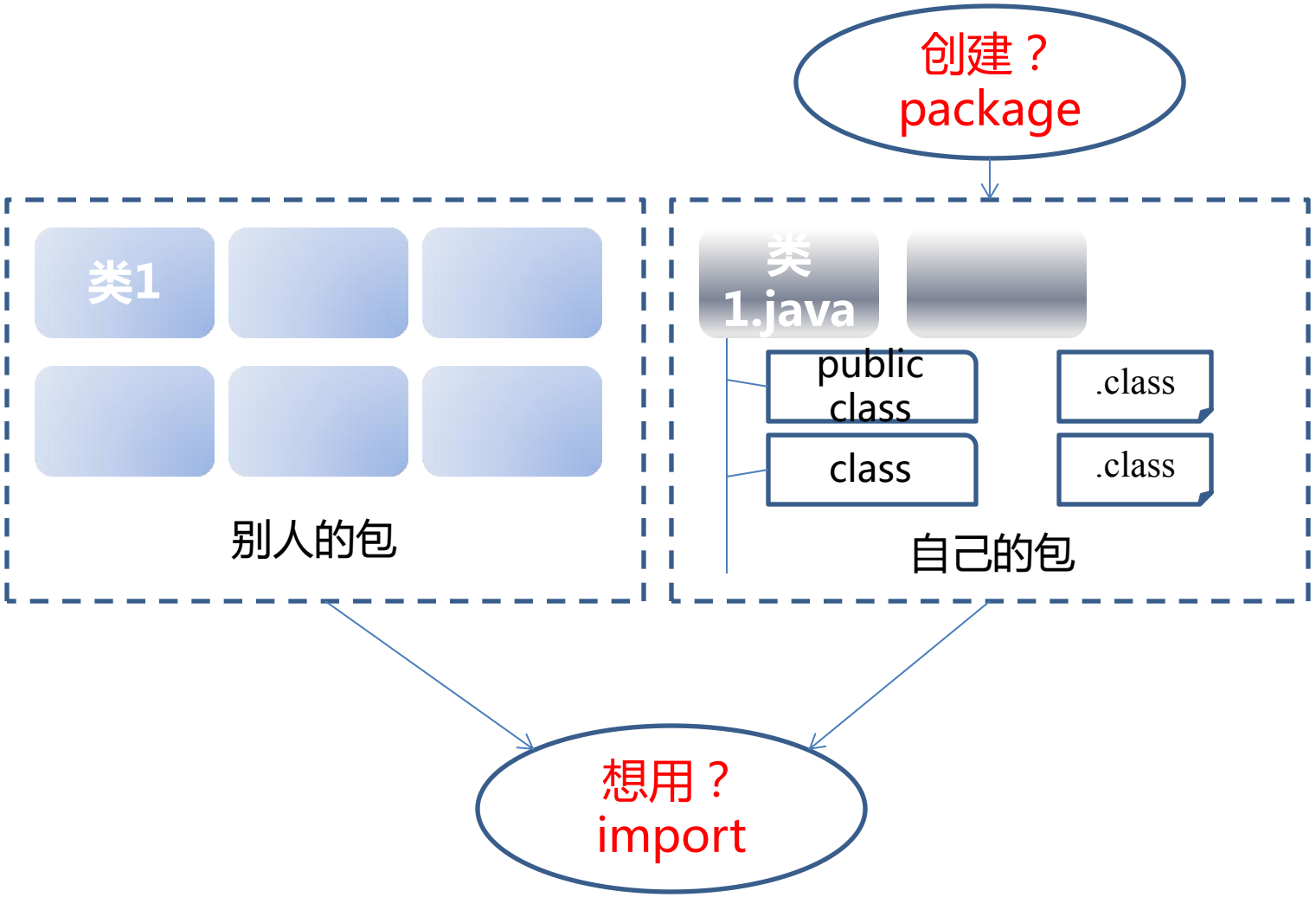
◆变量成员与方法成员



3.2 Java **Object**

- 3.2.1对象的创建
- 3.2.2对象的初始化
- 3.2.3成员初始化顺序
- 3.2.4垃圾回收机制

3.3包：库单元


创建？
package
类1
别人的包
类
1.java
public
class
class
.class
.class
自己的包
想用？
import

3.4 Java标准类库

- 3.4.1 Java.lang包和Java.util包
- 3.4.2 Object类
- 3.4.3 System类
- 3.4.4 Math类
- 3.4.5 String类、StringBuffer类、StringBuilder类

总结

◆面向对象程序设计的基本思想、提出的原因。

◆Java类是对象的抽象，类的定义，包括变量成员与方法成员、构造方法、finalize方法、以及访问控制符的作用。

◆一切都是对象，理解对象的创建、初始化、以及垃圾回收机制。

◆Java代码复用的机制，Java组织文件的方式--命名空间。

◆Java标准类库、Java常用的包、类和方法。

Enjoy your