

java_6_7_对象和类

内容：

类，对象，方法...

最早期，数据是无时代的管理，慢慢发展有了数组管理（数组将同一种类型数据放置在一起）（或是交给专门处理数据的软件管理，如后期专门学习的数据库[mysql,oracle]），继续发展，发现数组管理也有一定的局限性，只限定同一个类型之下插入数据值，后期发展会发现，数组没法满足现实世界中对于各种更多**不同类型**数据进行大量使用及存储（如姓名，年龄，地址，身份证号等等不同类型数据），为便于协作及管理，将相关数据和相关方法**封装**到一个独立的**实体**中，这时则出现了使用**对象**进行管理。

如姓名李四，年龄18，这些人具有的特征，就属于类的**属性**（或静态特征），而吃饭睡觉学习，这些则属于方法（动态行为），用世界上现实存在的物体用Java语言——对应的进行虚拟表现出来：

```
public class Person {  
    String name = "李四";  
    int age = 18;  
    String address = "上海";  
  
    public void eat(String name){  
        System.out.println(name + "早上八点吃饭");  
    }  
  
    public void sleep(String name){  
        System.out.println(name + "晚上十点睡觉");  
    }  
}
```

类的属性

Person类的两个方法

过程：无管理 - 》数组管理 - 》对象管理

*：封装类后期讲到，是一个**高内聚低耦合**，软件设计中通常用耦合度和内聚度作为衡量模块独立程度的标准，划分模块的一个准则是高内聚低耦合。从模块粒度来看，高内聚：尽可能类的每个成员方法只完成一件事（最大限度的聚合）；低耦合：减少类内部，一个成员方法调用另一个成员方法

Java 对象和类

面向对象

类，对象，方法，重载，实例，继承，封装，多态，抽象

1. 对象：对象就是类的一个实例[new]。
2. 类：类是一个模板，它描述一类对象的行为和状态。

对象和类的概念：

- **类(class)**：类就是一个模板（或一张图纸，只有先有图纸才能造出如汽车轮船笔记本...），根据类的定义造出来一个对象（因此先有类才有对象），它描述一类对象的行为和状态，类叫做**Class**
- **对象(object)**：对象是类的一个实例(类对象或类实例，都为同一个)，有状态和行为。如，汽车是一个对象，它的状态有：名称、品牌、颜色、价格...；行为(方法)有：开、跑...

*:程序开发平时会说，赶紧 **new** 个对象出来，没有对象是无法调用。

通过不断把公共部份抽象出来，会发现更多的属性及方法，如：某班级为一个类class，而具体的每个人为该类的对象 object 实例

类：某班级所有人
对象：李四
方法：吃饭，游玩，睡觉

Java中的对象

深入了解什么是对象，如果用心看一下我们周围真实的世界中，就会发现身边有很多对象，如，公交车，人等等种类 ... 所有这些对象都有自己特定的状态和行为。

如人它状态名称会有：名字、年龄；行为则有：吃，睡...

对比现实对象和软件对象，有很多的相似之处。

软件对象也可以表述出特定的状态和行为。软件对象的**状态**我们称之为**属性**，行为通过**方法**来体现。

属性用于**定义**该类(或该对象)包含的数据（静态特征），属性**作用**于整个类体中。

软件开发中，方法可以操作对象内部状态的改变，对象的相互调用也是通过方法来完成。（前面写的不少static void showXXX()...等等，就称之为类的方法，可以在main方法中直接调用）

Java中任何类的最高类(根类)都是指向 **Object**。

Java中的类

类可以看成是创建Java对象的模板（描述现实世界中存在的而又需要Java编写显示，使用定义的对象，一个**名称**对应一个**对象**，一个**字段名称**对应一个**对象属性**）。

创建一个简单的类,理解Java中类的定义,代码如下：

```
/**
 * 创建一个类，对象
 * @author djy
 * @version
 */
public class Person {
    //四个属性 ( field ) ,也叫定义成员变量
    String name;
    String address;
```

```

int age;

public static String IdCard;//身份证号

//三个方法method(eating,hungry,sleeping)
//吃
public void eating() {
    System.out.println("吃饭了...");
}

//饿
public void hungry() {
    String food = "鱼香肉丝";//局部变量
    System.out.println("张三饿了，赶紧吃" + food);
}

//睡觉
public void sleeping() {
    System.out.println("要睡觉，别打扰了");
}

//程序执行入口（如果要创建对象查看类的属性及方法，记得必须得有main主入口，方可查看编写效果是否可行）
public static void main(String[] args) {
    //创建一个对象，实例化无参的构造方法
    Person person = new Person();
    //访问（调用类中的两个方法）
    person.hungry();
    person.eating();
}
}

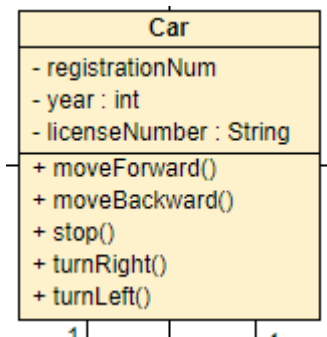
```

上面的Person类包含以下的类型变量：

- **局部变量**：在方法、构造方法或者语句块中定义的变量名称（如：String food = "鱼香肉丝"...）被称为局部变量。变量声明和初始化都是在方法中，方法结束后，变量自动销毁。
- **成员变量**：成员变量（如：name, age...）是定义在类中，方法体之外的变量。这种变量在创建对象的时候实例化。成员变量可以被类中的方法、构造方法和特定类的语句块调用访问或是设置值。
- **类变量**：类变量（如上面的static String IdCard）也声明在类中，方法体之外，但必须声明为static类型。

一个类可以拥有多个方法，在上面的例子中：eating(),hungry(),sleeping()都是class类（Person）的方法。

UML类图（Unified Modeling Language,统一建模语言或标准建模语言,了解知道类图方式，企业工作中使用PowerDesigner画对应类图）



构造方法

每个类都有自己的**构造方法**。如果没有显式地为类定义构造方法，Java编译器则会为该类提供一个**默认**的**无参**构造方法（`new Person();`）。

在创建一个对象的时候，至少要调用一个构造方法。构造方法的名称必须与类的名称**一样**，一个类可以有多个构造方法（无参或是不同的参数）。

创建构造方法，代码编写如下：

```
/**
 * 男孩
 * @author djy
 * @version
 */
public class Boy {
    String name;
    String address;
    int age;

    // 无参构造方法
    public Boy(){
    }

    //有一个参数(name)的构造方法
    public Boy(String name){
        this.name = name;
        System.out.println("人的名字叫：" + name);
    }

    // 两个参数(name及age)的构造方法
    public Boy(String name,int age){
        this.name = name;
        this.age = age;
        System.out.println("人的名字叫：" + name + "，年龄：" + age + "岁了");
    }

    public void sleeping() {
        System.out.println("睡觉了");
    }

    public void sleeping(String name) {
```

```

        System.out.println(name + "睡觉了");
    }

    //程序执行入口（如果要创建对象查看类的属性及方法，记得必须得有main主入口，方可查看编写效果是否可行）
    public static void main(String[] args) {
        //创建一个对象，实例化无参的构造方法
        Boy boy = new Boy();
        boy.address = "上海"; // 给字段属性赋值
        //访问Boy类中睡的方法
        boy.sleeping();

        //创建一个对象，实例化带 name字段属性的构造方法
        Boy boy2 = new Boy("张三");
        boy2.sleeping("张三");
    }
}

```

要点：

1. 通过new关键字创建对象，调用对象
2. 构造器[或叫构造方法]不能定义返回值类型（返回值类型是本类），不能在构造器中使用return返回某个值
3. 如果没有定义构造器，系统编译器自动给我们定义一个无参的构造器，如果已定义，编译器则不会再自动添加
4. 构造器方法名必须与类名一样，否则就变成为类的一个方法
5. 如果添加了带参数的构造器，则必须要手动添加上无参的构造器

方法重载Overload

方法重载是指在一个类中定义多个**同名方法**，但要求每个方法具有**不同参数类型或参数个数**。

Java的方法重载，就是在类中可以创建多个方法，它们可以有相同的名字，但必须具有**不同的参数**，即或者是参数的**个数不同**，或者是参数的类型不同。调用方法时通过传递给它们的不同个数和类型的参数来决定具体使用哪个方法。

```

Boy boy2 = new Boy();
boy2.sleeping(); //不带参数的方法
boy2.sleeping(name); //带一个参数方法
//更多参数，类型的方法...

```

*：重载和重写的区别（接口方法被实现是重写 或 重写Object的toString, public String toString() {return "@对象地址";}）

创建对象

对象是根据类创建的,使用关键字**new**创建一个新的对象。创建对象过程：

1. 声明一个对象，分配对象空间，并将对象的成员变量初始化为0或是null空字符串
2. 执行属性值显示初始化
3. 执行构造（无参或带参）方法
4. 返回一个对象地址给相关变量

创建对象代码编写（new方式）：

```
public static void main(String[] args) {  
    String name = "张三";  
    int age = 18;  
    //Boy boy = new Boy(); //无参  
    Boy boy2 = new Boy(name, age); //带两个参数构造器  
    //至此对象地址@15db9742已生成  
}
```

访问实例变量和方法

通过已创建的对象来访问成员变量和成员方法，如下所示：

```
//实例化对象  
Boy boy2 = new Boy();  
//访问类中的方法  
boy2.sleeping(name);  
//设置成员变量(给年龄赋值20岁)  
boy2.setAge(20);  
//访问成员变量(获取年龄的值)  
int ageVal = boy2.getAge(); //手动写set, get方法
```

实例化对象(new对象),访问实例对象中的变量和调用方法，代码如下：

```
public static void main(String[] args) {  
    String name = "李四";  
    int age = 20;  
    Boy boy2 = new Boy();  
    boy2.setAge(age); //给年龄赋值  
    System.out.println(name + "的年龄为：" + boy2.getAge() + "岁了");  
}
```

虚拟机内存三个区域

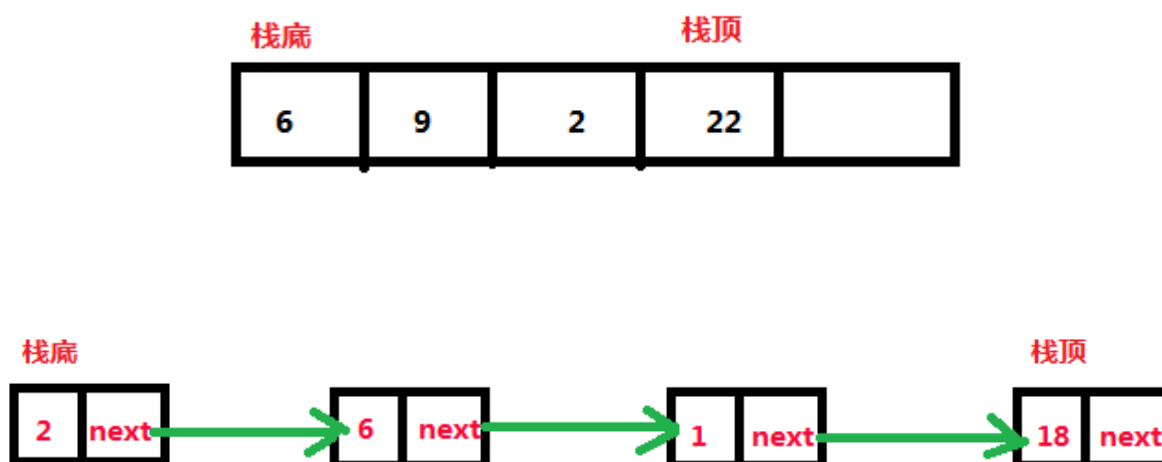
此块了解,结合断点一起对比使用。

栈 (stack)

特点：

1. 栈描述的是方法执行时的内存模型，每个方法在被调用时都会创建一个栈帧（存储局部变量，操作数，方法出口等）
2. JVM为每个线程创建一个栈，用于存放此线程执行方法的信息（实际参数，局部变量等）
3. 栈属性线程中私有（main程序中的主线程，是唯一一个公有的），不能实现线程之间共享
4. 栈的存储特征是"先进后出，后进先出"的执行顺序（如平时看到羽毛球放置盒），也简称为FILO(全称英文：First In Last Out), 最早进入的元素存储的位置叫作栈底（bottom），最后进入的元素存放的位置叫作栈顶（top）
5. 系统自动分配栈空间，速度快，栈是一个连续的内存空间

栈这种数据结构即可以用数组（下图第一个）来实现，也可以用链表（下图第二个）来实现：



入栈（push）：把新元素放入栈中，只允许从栈顶的一侧放入元素，新元素的位置将成为新的栈顶

出栈（pop）：把元素从栈中弹出，只有栈顶元素才允许出栈，出栈元素的前一个元素将成为新的栈顶

入栈出栈只会影响到最后一个元素，不涉及其它元素的整体移动，速度快。

栈也相当于，系统自动分配的一个个小小的操作**工作台**。

堆(heap)

特点：

1. 堆主要用于存储创建好的对象和数组
2. JVM只有一个堆，被所有的线程共享
3. 堆是一个不连续的内存空间，分配灵活，速度慢

方法区(method area)

方法区（或叫静态区）特点：不变方面

1. JVM只有一个方法区，被所有线程共享
2. 方法区实际上也是堆（堆中有方法区），只是用于存储类，常量相关信息
3. 用来存放程序中永远不变或是唯一的内容（如类信息中的class对象，静态变量，字符串常量等）

调用输出对象类地址：

```
//程序执行执行的入口，要是测试则必须要有main
```

```

public static void main(String[] args) {
    //创建一个对象，实例化无参的构造方法
    //Boy boy = new Boy();
    //System.out.println(boy);// @2a139a55
    //boy.address = "苏州";// 给字段属性赋值
    //boy.sleeping();

    // 创建一个对象//实例带new Boy(name)字段属性的构造方法
    Boy boy2 = new Boy();
    boy2.name = "李四";
    boy2.age = 18;// 给字段属性赋值
    boy2.address = "苏州";// 给字段属性赋值
    //System.out.println(boy2);// 输出：@15db9742
    boy2.sleeping(boy2.name);
}

```

执行输出发现@加数字，则是创建对象实例化的存储的一个**地址**，如果判定是不是同一个对象，查看此地址即知：

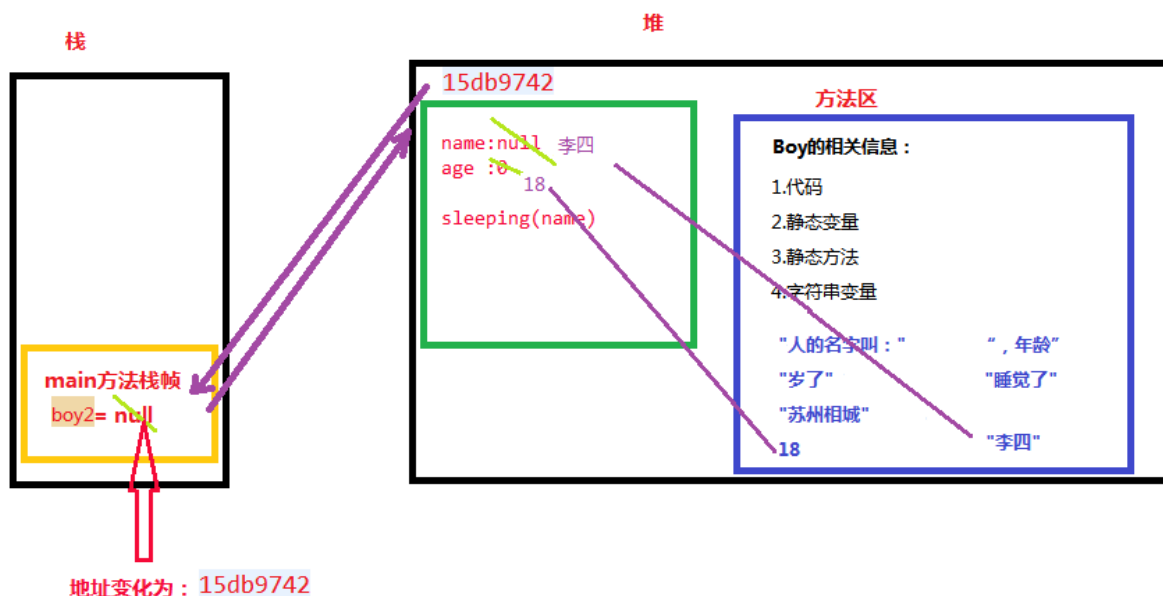
```

Boy@2a139a55
....
Boy@15db9742
....

```

此处则存在两个对象。

则开始第一次时，栈的Boy对象是null,当一步步执行（或是调试查看过程），有了@15db9742对象地址等等，如图：



依据地址进行相关联。

this

this:是指创建好的对象的地址，表示已创建好的对象（由于构造方法在调用前，对象就已经创建好，因此使用this则是指向于当前对象）

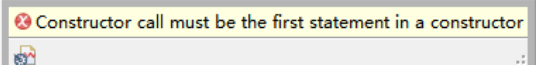

```
//一个参数(name)的构造方法
public Boy(String name){
    //super();//构造方法的第一句总是super()//如果不加，编译器自动帮我们加上，但不可以放在第二行以下
    this.name = name;//this则表示创建好的对象
}
```

this也可以调用带参的构造器，但必须放置在第一行：

```
// 两个参数(name及age)的构造方法
public Boy(String name,int age){
    this.name = name;
    this.age = age;
    System.out.println("人的名字叫: "+name+",年龄: "+age+"岁了");
}

public Boy(String name, String address, int age) {
    //this(name, age);
    this.address = address;
    this(name, age);
}
```

如果使用this调用前面带参数的构造方法，this则必须放置在第一行



注：this可以调用方法（this.sleeping(name);）或静态方法或构造方法(this(name, age);），但static无法调用方法（static处说明）

this中常用用法：

- 1.this用来指明当前对象；普通方法中，this总是指向调用该方法的对象；构造方法中，this则是指向正要初始化的对象
- 2.使用this关键字调用重载的构造方法，避免相同初始化代码（代码简洁），但只能在构造方法中用，且必须是在构造方法中的第一行
- 3.this不能用于static方法中

static

类中使用static声明过的成员变量则称为静态成员变量，也叫类变量，类变量的生命周期与类一样，在整个应用程序执行期间都是有效

static修饰过的成员变量和方法，都从属于类，普通变量和方法从属于对象（因此对象new创建成功，普通变量和方法才产生）

```
//静态成员变量
public static String IdCard;//身份证号

public static void showIdCard(){//静态方法
    //sleeping();//调用非静态成员，编译器报错
    //静态方法调用非静态方法则会提示：Cannot make a static reference to the non-static
method sleeping() from the type Person
    //调用非静态成员或非静态方法编译器报错原因：静态方法相当于只有图纸，不一定可以造出房子，但如果有房子对象，就必然有图纸，因此sleeping方法调用静态方法却没有问题，反之不行
    System.out.println(IdCard);
}
```

```
public void sleeping() {
    showIdCard();//普通方法调用静态成员变量或是静态方法都是可以的
    System.out.println(IdCard);

    System.out.println("睡觉了");
}
```

堆方法区：



静态初始化块

静态初始化方法，用于类的初始化操作，在静态初始化块中不能直接访问非静态成员（构造方法用于对象初始化，类一旦创建就直接生成而且程序中只会调用一次）

了解：

1. 追溯至Object类（源码查看），先执行子类的静态初始块，直到我们自己创建的类的静态初始化块为止
2. 构造方法执行与1一样

```
/**
 * 静态初始化块（静态代码块）
 * @author Administrator
 * @date 2019年7月10日
 * @version
 */
public class DemoStatic {

    static String dbName = "root";//账号名称
    static String dbPwd = "root";//password: pwd 密码

    static{
        System.out.println("欢迎执行...");
        System.out.println("账号: "+dbName+", 密码: "+dbPwd);
    }

    public static void main(String[] args) {
        DemoStatic demoStatic = new DemoStatic();
        //Object
    }

}
```

静态初始化块相当于仅仅只有图纸而没有调用如吃方法等

object源码：

```
7 public class Object {
8
9     private static native void registerNatives();
10    static {
11        registerNatives();
12    }
13 }
```

值传递

Java方法中所有的参数都为“值传递”，只是“传递的是值的副本”，相当于原件（原参数）只有一个，复印却是可以多次，而不影响原件的影响。

基本数据类型参数的值传递：传递值的副本，不影响原件

引用类型参数的值传递：传递值的副本，但引用类型指的是“对象地址”（本身引用类型就是对象），因此，副本和原参数都指向**同一个“地址”**，若改变副本指向地址对象的值，那么意味着原参数指向对象的值也一并发生了改变。

```
public class User2 {
    String username;

    public User2(String username) {
        this.username = username;
    }

    public void showName(User2 u){
        //System.out.println(u2);//@2a139a55
        u.username = "赵六";
    }

    public static void main(String[] args) {
        User2 u2 = new User2("李四");
        //System.out.println(u2);//@2a139a55

        //System.out.println(u2.username);//原来u2.username为李四
        u2.showName(u2);//然后把前面的值覆盖后，则变成赵六
        //当调用showName(对象)时，把对象往下传，传入的也是同一个@2a139a55地址
        System.out.println(u2.username);
    }
}
```

指向都是同一个对象，因此值改变（调试更清晰）。

说明：

- 1). u2指向地址：2a139a55 username为李四
- 2). 调用showName(xx)方法时，u的地址指向还是@2a139a55,此时username赋值为李五
- 3). 因此当System.out.println(u2.username)时输出的已是赋过之后的值赵六

注：最初原本和副本username的值都是李四，当执行到u.username = "赵六";此行，原先的值已被后面的覆盖，username重新给值"赵六"

```
public class User2 {
    String username;

    public User2(String username) {
        this.username = username;
    }

    public void showName(User2 u){
        //System.out.println(u);//@2a139a55
        u.username = "赵六";
    }

    public void showName2(User2 u){
```

```

        u = new User2("李六");//new已经是创建新的对象
        //System.out.println(u.username);
        //System.out.println(u);//@15db9742
    }

    public static void main(String[] args) {
        User2 u2 = new User2("李四", "123456");
        //System.out.println(u2);//@2a139a55

        //System.out.println(u2.username);//原来u2.username为李四
        u2.showName(u2);//然后把前面的值覆盖后，则变成赵六
        //当调用showName(对象)时，把对象往下传，传入的也是同一个@2a139a55地址
        System.out.println(u2.username);

        System.out.println("=====");
        u2.showName2(u2);
        //System.out.println(u2);//@2a139a55
        System.out.println(u2.username);

    }
}

```

断点代码调看。

说明：

- 1). u2指向地址：2a139a55 username为李四
- 2). 调用showName(xx)方法时，u的地址指向还是@2a139a55,此时username赋值为赵六
- 3). 当System.out.println(u2.username)时输出的已是赋过之后的值赵六
- 4). 当调用showName2(xx)方法时（前面showName局部u对象已不存在）的u指向的地址为@15db9742，username值是李六
- 5). 当System.out.println(u2.username)时却会发现，输出的显然还是赵六不是李六（此处的u2指向的地址还是@2a139a55，地址并没有改变，还是同一个对象）

GC（了解）

Java引用垃圾回收机制，全称Garbage Collection 垃圾收集。

引入GC,让Java程序员可以将更多的精力放到业务逻辑处理中，而不需考虑内存管理。

GC（相当于饭店服务员，帮忙回收吃剩的垃圾）

GC回收过程只做两件事情：

1. 发现无用的对象
2. 回收无用对象占用内存空间

GC主要是将“无用对象”进行回收，无用的对象就是没有任何变量引用此对象，处于完全空置状态，GC通过一些算法方式，发现无用对象，然后进行清除和整理。

相关算法：计数法

相关算法：

计数法：

给对象中添加一个引用计数器，每当有一个地方引用它时，计数器值就加1；当引用失效时，计数器值就减1；任何时刻计数器为0的对象就是不可能再被使用的

可达性分析（以据搜索算法）：

从根（GC Roots）的对象作为起始点，开始向下搜索，搜索所走过的路径称为“引用链”，当一个对象到GC Roots没有任何引用链相连（用图论的概念来讲，就是从GC Roots到这个对象不可达）时，则证明此对象是不可用的。

更多了解（如分代垃圾）参考：<https://www.cnblogs.com/smyhvae/p/4744233.html>

heap堆可达分析：

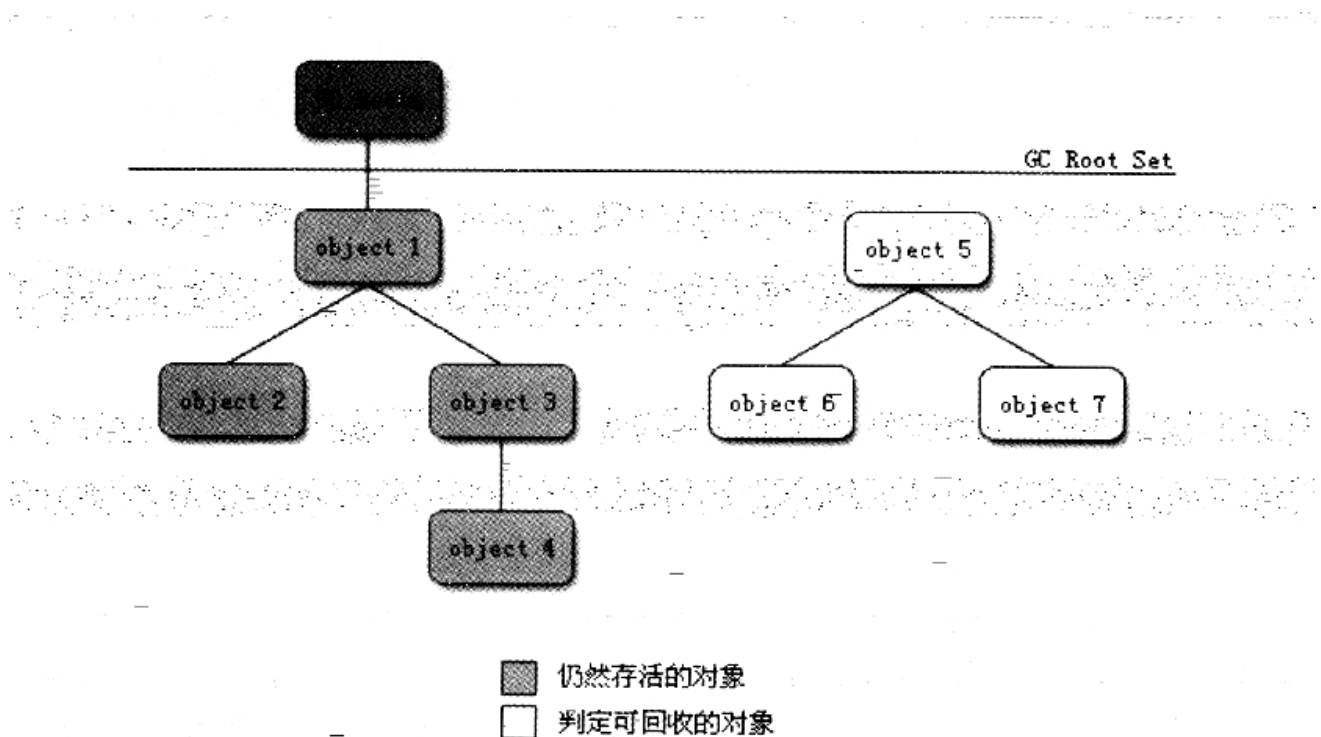


图 3-1F 可达性分析算法判定对象是否可回收

package包

通过package来实现对项目开发中类的管理。

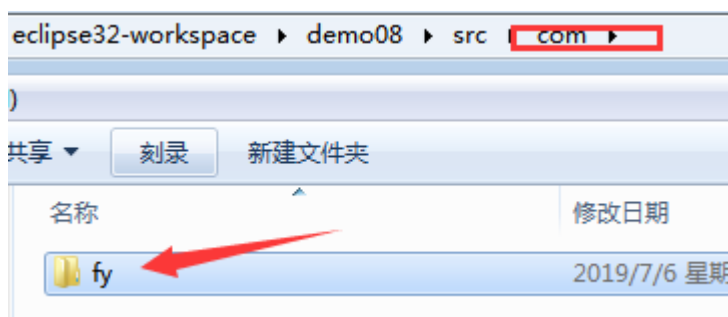
相当于不同省下市镇等等层级关系下的**地址门牌号**，因此Java中也需要使用包来进行编写我们的项目代码，方便管理及引用查找（相当于电脑中有各种各样的**文件夹**，需要分门别类）。

包主要用来对类和接口...进行分类,当开发程序越写越多时，编写成百上千的类，这时候就要我们所写的类和接口进行分类,尽量编写有**意义**的包名称来进行**识别**。

两个特点：

- 1).package放置在第一行（非注释代码语句前）
- 2).包名：使用域名倒序写，加模块名，便于内部管理类，如:com.baidu.user或com.jd.log

注：创建包并查看是否对应自动放置在类的第一行



相当于也是存储的是一个文件夹，**域名倒序写**，是为了不会有重名，因此创建包时，注意**不要有重名**。

jdk本身也有很多的包管理，后期引用更多类时，每个类都是有包的机制（除了java.lang下的包不需导入）。

Import

Java中，给出一个完整的限定名，包括包名、类名，Java编译器就可以很容易定位到源代码或者我们编写的类。Import就是用来提供一个合理的路径，使得编译器可以找到这个类。引用方式：

导入包的快捷键：**Alt+I**

```
package com.fy.test;

import java.util.Date;
//import java.sql.Date; //与前面的重名，同时写入不知引用的是那个包下的Date类
import com.fy.model.User; //导入User类
import com.fy.model.*; //导入model下所有的类，*加载所有，会降低编译运行速度
/**
 * import导入方式
 * @author djy
 * @date 2019年7月6日
 * @version
 */
public class UserTest {

    public static void main(String[] args) {
        User u1 = new User();
        //com.fy.model.User u2 = new com.fy.model.User();
        System.out.println(new Date());
    }

}

//先创建com.fy.model包，创建User
```

商品样例

```

/**
 * 商品列表
 * @author Administrator
 * @date 2019年1月13日
 */
public class ShopItem {
    String name;
    double price;
    int num;//库存数量

    public void setNum(int num){
        this.num = num;
    }

    public void setName(String name){
        this.name = name;
    }

    public void setPrice(double price){
        this.price = price;
    }

    public void showShopItem() {
        System.out.println("商品名称为："+name);
        System.out.println("商品价格为："+price);
        System.out.println("商品库存还有："+num);
    }
}

```

new对象，调用方法：

```

public class ShowShop {

    public static void main(String[] args) {
        ShopItem shopItem = new ShopItem();
        shopItem.setName("联想笔记本");
        shopItem.setPrice(4000);
        shopItem.setNum(300);

        shopItem.showShopItem();
    }
}

```

运行结果：

```

商品名称为：联想笔记本
商品价格为：4000.0
商品库存还有：300

```