Lesson3
认识对象：封装数据为类

主讲老师：申雪萍

主要内容

- 类的设计原则
- Java程序的基本结构
- 类的定义
  - 成员变量的定义（实例变量和类变量）
  - 成员方法的定义（实例方法和类方法）
  - 方法重载
- 对象的生成、使用和清除
  - 默认构造函数（不带参数构造函数）
  - 带参数构造函数
  - 垃圾内存自动回收机制
- 匿名对象
- 进一步理解引用类型变量
- 类变量和类方法

类的设计原则

一. 取有意义的名字。

二. 尽量将数据设计为私有属性。

三. 尽量对变量进行初始化。

四. 类的功能尽量单一（原子、细粒度）。

源文件布局

- Java 源文件的基本语法：

```
[<包声明>]
   [<导入声明>]
   <类声明>+
```

- 示例，VehicleCapacityReport.java 文件：

```
package shipping.reports;

import shipping.domain.*;
import java.util.List;
import java.io.*;

public class VehicleCapacityReport {
   private List  vehicles;
   public void generateReport(Writer output) {...}
}
```

Java程序的基本结构

- 一个基本Java程序的三大件
  1、包的声明；//指定类所在的位置
  2、类的导入；//用到其它位置中的类
  3、类的定义；//核心部分

- Java允许在一个Java源文件中编写多个类，但其中的多个类至多只能有一个类使用public修饰。

Student.java（一个类放在一个源文件中）

```
package com.buaa.hasEx;
import java.util.*;
public class Student {
        // 成员属性：学号，姓名，性别，年龄
        private String student_id;
        private String student_name;
        private String student_sex;
        private int student_age;


        // 无参数构造
        public Student() {
                super();
        }
```

ClassTest.java（多个类放在一个源文件中））

```java
package com.buaa.classEx;
import java.util.*;
public class ClassTest{
    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        System.out.println("Hello World!!!");
    }
}
class Test1{

}
class Test2{

}
```

主要内容

- 类的设计原则
- Java程序的基本结构
- 类的定义
  - 成员变量的定义
  - 成员方法的定义
  - 方法重载
- 对象的生成、使用和清除
  - 默认构造函数（不带参数构造函数）
  - 带参数构造函数
  - 垃圾内存自动回收机制
- 进一步理解引用类型变量
- 类的静态属性和静态方法

定义一个类的步骤

一． 定义类名（每个单词的第一个字母大写，其余的小写）

二． 编写类的属性

三． 编写类的方法

类的定义（[]代表可有可无）

- 类的定义格式如下：

```
[类修饰符] class  类名  extends 基类
 implements 接口列表
 {
     [数据成员定义]
     [成员方法定义]
 }
```

关键字class表示类定义的开始

类名要符合标识符的命名规范

修饰符分为访问控制符和类型说明符

类的定义说明

一. 关键字class表示类定义的开始

二. 类名要符合标识符的命名规范

三. 修饰符分为访问控制符和类型说明符

访问控制符

- 类的访问控制符有两个，一个是public，即公共类，另一个就是默认，即没有访问控制符。
  - ① 一个类被定义为公共类，就表示它能够被其它所有的类访问和引用。
  - ② 在一个Java源程序中只能有一个public类，这个类一般含有main方法。
  - ③ 不用public定义的类，其只能被同一个包中定义的类访问和引用。
  - ④ 在一个JAVA程序中可以定义多个这样的类。

类型说明符

- 类的类型说明符主要有两个
  ① final
  ② abstract

成员变量的定义（[]代表可有可无）

- 成员变量的定义格式：
  - [修饰符] 变量的数据类型 变量名[=初始值]
- 修饰符主要有四种，分别是：
  ① this(大多数可以省略，实例变量)
  ② static(类变量)
  ③ public、
  ④ private、
  ⑤ protected、
  ⑥ 默认。

成员方法的定义（[]代表可有可无）

- 方法的定义格式：

```
[修饰符] 返回值类型 方法名([形参说明])
[throws 例    外名1,例外名2…]
 {
        局部变量声明;
        执行语句组;
 }
```

成员方法的定义说明

一. 常用的修饰符为public、private、protected、static、final等

二. 返回值类型：方法一般需要有一个返回值表示执行结果，也可以无返回值（用void表示）。返回值类型可以是Java类型系统中的所有类型。

三. 有返回值的方法使用return语句将值返回给调用者。

创建对象

一. 必须使用new关键字创建一个对象

二. 使用对象属性（对象名. 成员变量）

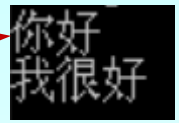
三. 使用对象方法（对象名. 方法名）

四. 同一个类的每个对象有不同的成员变量的存储空间

五. 同一个类的每个对象共享该类的方法

return的理解（特殊点强调）

```
class  Dog{
        String name;
        public  void bark( int  a){
        if(a==0){
                System.out.println(“你好”);
        }else if(a==1){
                return;
              }
        System.out.println(“我很好！”);
          }
}
public class FangFa1{
        public static void main(String[ ] args){
                Dog  A=new Dog();
                A.bark( 1 );
        }
}
```



结束方法，无输出

this引用

- Java 每个类都默认地具有 null、this、super三个域，所以在任何类中都可以不加说明就可以直接引用它们：

  1. null ：代表“空”，用在定义一个对象但尚未为其开辟内存空间时。

  2. this 和 super ：是常用的指代本类对象和父类对象的关键字

this关键字

- this可以看做一个变量，他的值是当前对象的引用
- 在类的方法定义中使用this关键字代表使用该方法的对象的引用
- 使用this可以处理方法中成员变量和局部变量重名的问题

this关键字

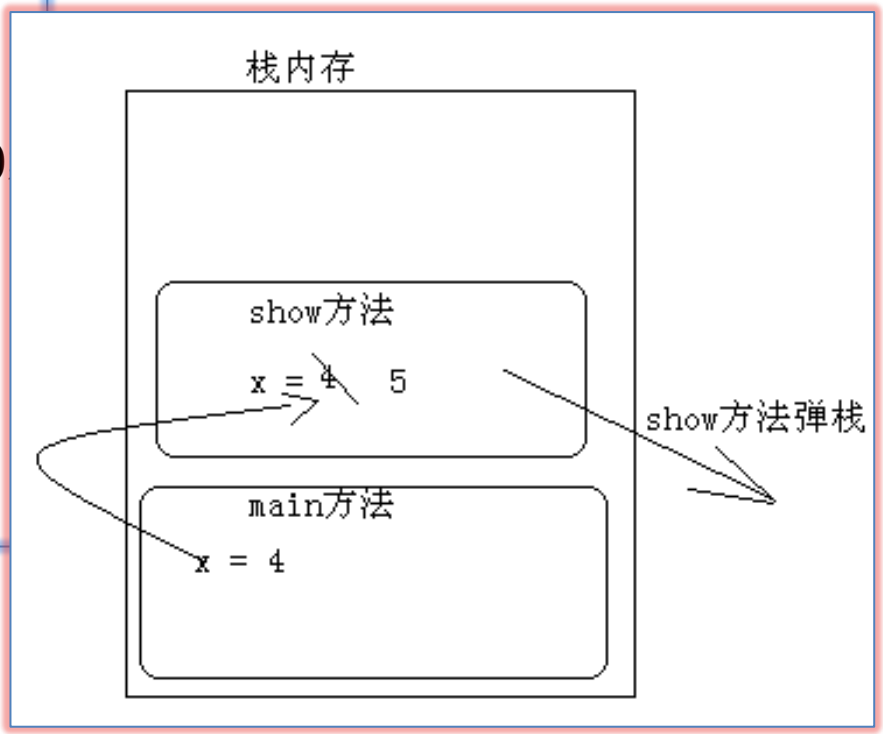
```
class  Cat{
        String name;
        String xb;
        int nl;
        public  void  set ( String name,String xb,int nl){
                this.name=name;
                this.xb=xb;
                this.nl=nl;
         }
        public String toString(){
                return nl+"岁的"+name+xb+"睡着了";
        }
}
public class Test{
        public static void main(String[] args){
                Cat mao=new Cat();
                mao.set("花花","先生",2);
                System.out.println(mao);
        }
}
```

成员变量与方法的参数同名

为了区分两个name，用this代表使用该方法的对象mao的引用

基本类型和引用类型作为参数传递

```
class Demo
{
        public static void main(String[] args)
        {
                int x = 4;
                show(x);
                System.out.println("x="+x)
        }
        public static void show(int x)
        {
                x = 5;
        }
}
```

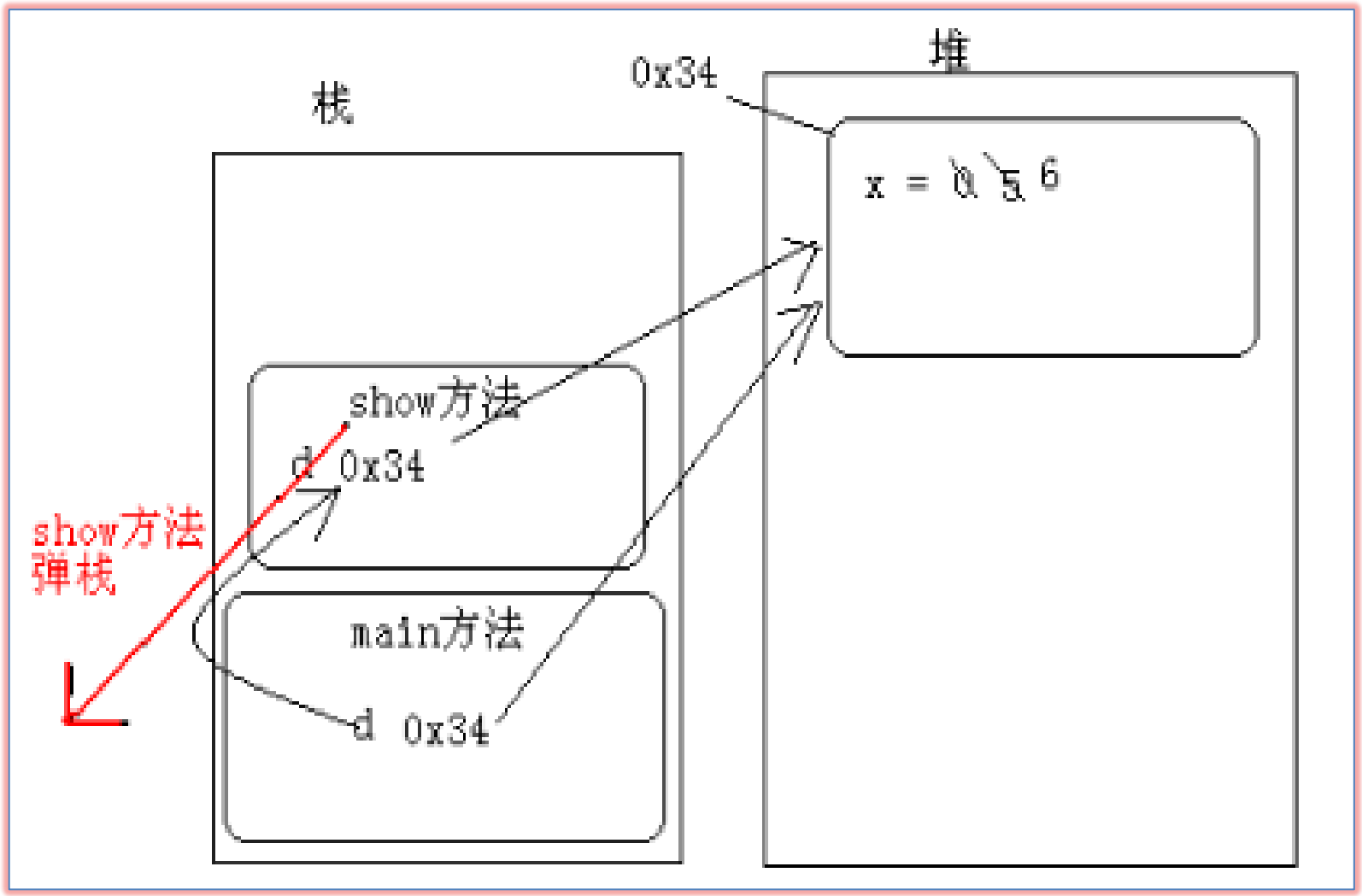


基本类型作为参数传递 （深拷贝）

- 基本类型作为参数传递时，其实就是将基本类型变量x空间中的值复制了一份传递给调用的方法show()，当在show()方法中x接受到了复制的值，再在show()方法中对x变量进行操作，这时只会影响到show中的x。当show方法执行完成，弹栈后，程序又回到main方法执行，main方法中的x值还是原来的值。

当引用变量作为参数传递时 （浅拷贝）

```
class Demo
{
        int x ;
        public static void main(String[] args)
        {

                Demo d = new Demo();
                d.x = 5;
                show(d);
                System.out.println("x="+d.x);
        }
        public static void show(Demo d)
        {
                d.x = 6;
        }
}
```


堆
0x34
栈
x = 5 6
show方法
d 0x34
show方法
弹栈
main方法
d 0x34

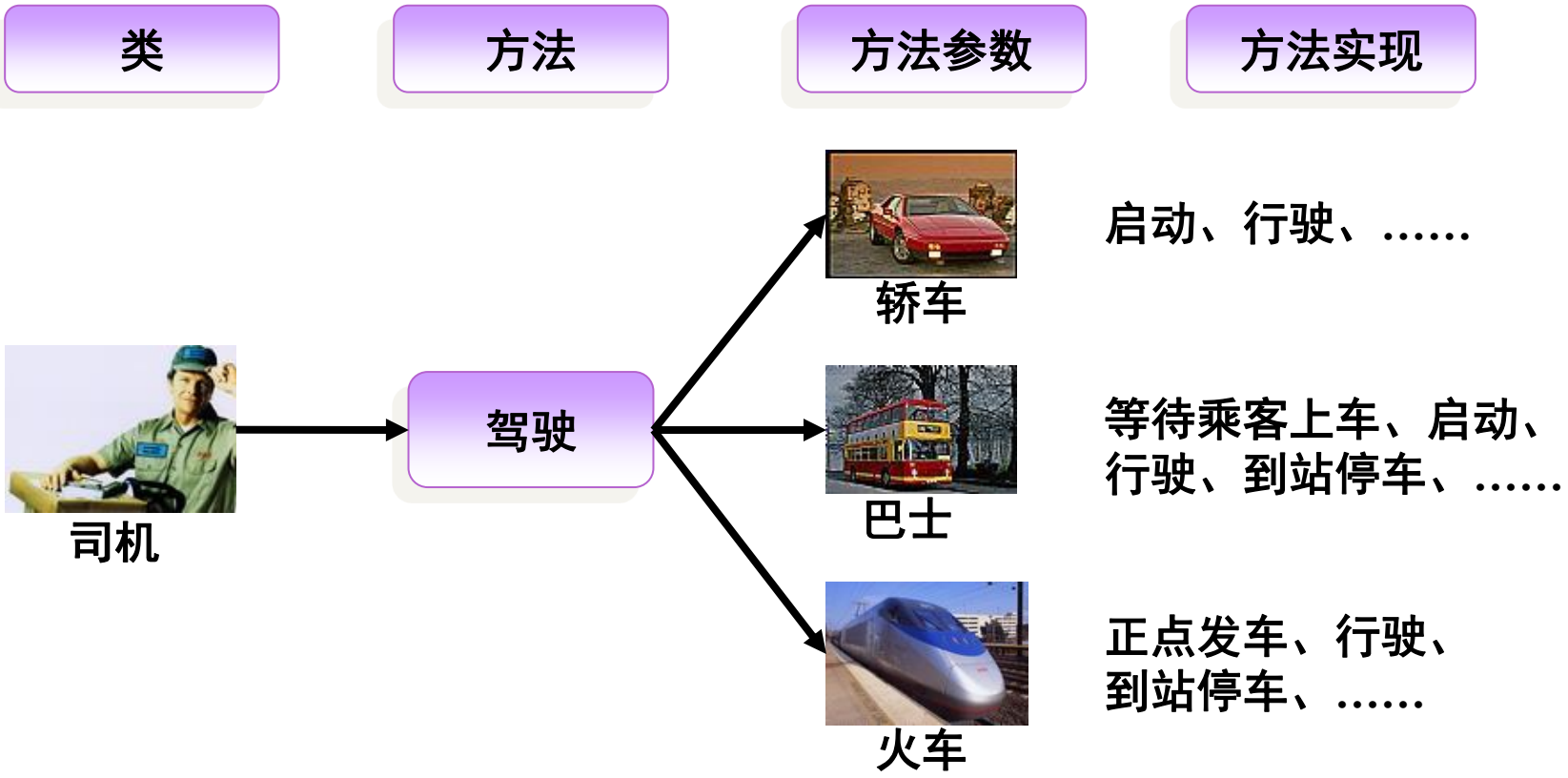
当引用变量作为参数传递时

- 当引用变量作为参数传递时，这时其实是将引用变量空间中的内存地址(引用)复制了一份传递给了show方法的d引用变量。这时会有两个引用同时指向堆中的同一个对象。当执行show方法中的d.x=6时，会根据d所持有的引用找到堆中的对象，并将其x属性的值改为6.。
- 由于是两个引用指向同一个对象，不管是哪一个引用改变了引用的所指向的对象的中的值，其他引用再次使用都是改变后的值。

方法的重载

- 方法的重载是指一个类中可以定义有相同的名字，但参数不同的多个方法，调用时会根据不同的参数表选择对应的方法。

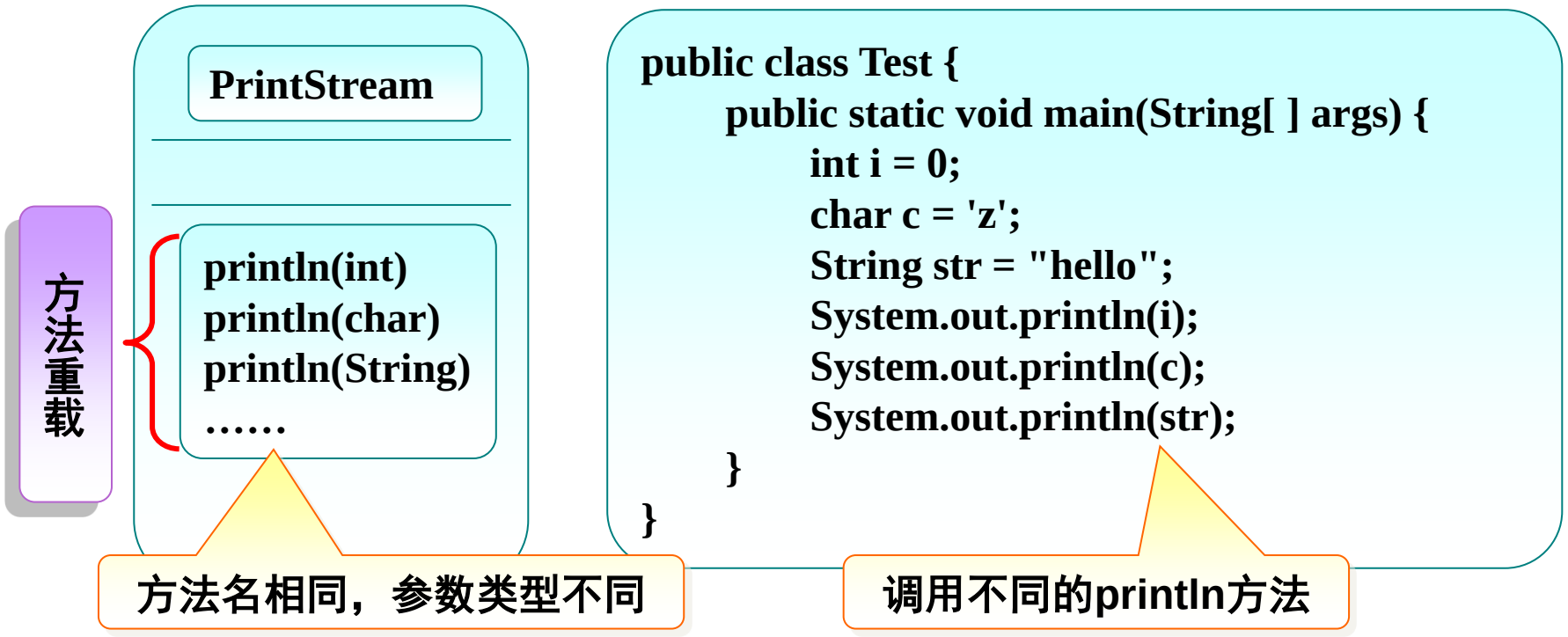
生活中的方法重载



如果用代码实现，我们需要三个方法，这些方法的方法名称相同，参数类型不同

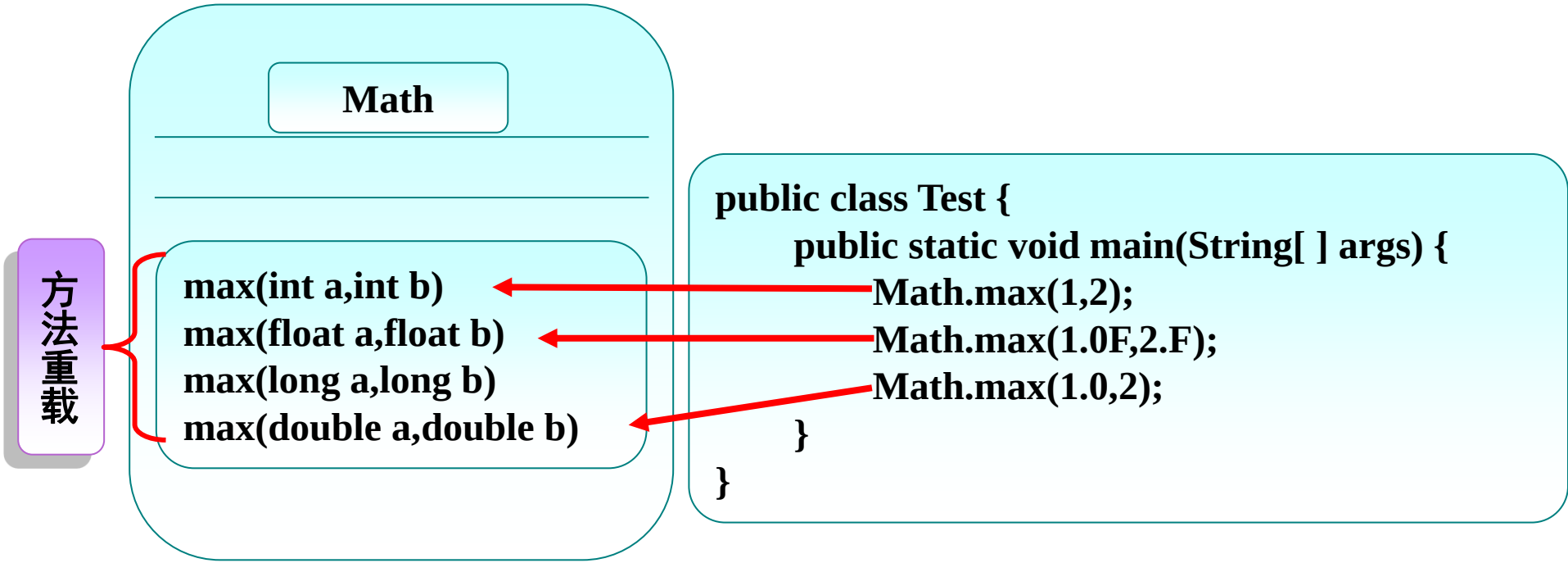
方法重载的代码示例

- java.io.PrintStream类的println方法能够打印数据并换行，根据数据类型的不同，有多种实现方式



方法重载的代码示例

- java.lang.Math类的max()方法能够从两个数字中取出最大值，它有多种实现方式
- 运行时，Java虚拟机先判断给定参数的类型，然后决定到底执行哪个max()方法



主要内容

- 类的设计原则
- Java程序的基本结构
- 类的定义
  - 成员变量的定义
  - 成员方法的定义
  - 方法重载
- **对象的生成、使用和清除**
  - **默认构造函数（不带参数构造函数）**
  - **带参数构造函数**
  - **垃圾内存自动回收机制**
- 匿名对象
- 进一步理解引用类型变量
- 类的静态属性和静态方法

构造函数(Constructor)

一．构造方法(constructor)是一类特殊的成员方法。

二．从功能上讲，它使用new关键字用来对新创建的对象进行初始化的。

三．从形式上来讲，它有以下特点：

① 它与类同名；

② 它没有任何返回值；

③ 除了上述两点外，在语法结构上与一般的方法相同。

构造方法重载

- Java语言中，每个类都至少有一个构造方法；

- 构造方法有两大类，构造方法可以重载，并且通常是重载的：
  - 不带参数构造方法（**默认构造方法**）
  - 带参数构造方法

为什么需要构造方法

```
public class Teacher4 {
        private String name;         // 姓名
        private int  age;            // 年龄
        private String education; // 学历
        private String position;    // 职位

        public String introduction() {
                return "大家好！我是" + name + "，我今年” + age + "岁，学历“
                                                +education+"，目前职位是"+position;
        }
        public String getName() {
                return name;
        }
        public void setName(String myName) {
                name = myName;
        }
        // 以下是其他属性的setter、getter方法，此处省略
}
```

为什么需要构造方法

```
public class Teacher4Test {
    public static void main(String[ ] args) {
        Teacher4  teacher = new Teacher4();
        teacher.setName(“Mary");
        teacher.setAge(23);
        teacher.setEducation("本科");
        System.out.println(teacher.introduc
        tion());
    }
}
```

Teacher4中有太多的属性及对应的setter方法，在初始化时，**很容易就忘记了，造成代码冗余**，有没有可能简化对象初始化的代码？

大家好！我是Mary,我今年23岁，学历本科，目前职位是null

要简化对象初始化的代码，可以通过构造方法来解决

不带参数构造方法（默认构造方法）

```
public class Teacher5 {
        private String name;  // 教员姓名
        // 构造方法
        public  Teacher5() {
                name = “Mary";
        }
}
```

```
public class Teacher5Test {
        public static void main(String[ ] args) {
                Teacher5  teacher = new Teacher5();
        }
}
```

带参数的构造方法

- 通过带参数的构造方法，显式地为实例变量赋予初始值

带参数的构造方法

```
public class Teacher6 {
    private String name;        // 教员姓名
    private int age;            // 年龄
    private String education;   // 学历
    private String position;    // 职位
    // 带参数的构造方法
    public Teacher6(String pName,int pAge,String pEducation,String
                                                        pPosition) {
        name = pName;
        age = pAge;    // 可以增加对age等属性的存取限制条件
        education = pEducation;
        position = pPosition;
    }
    public String introduction() {
        return "大家好！我是" + name + "，我今年" + age + "岁，学历" +
        education + "，目前职位是"+position;
    }
}
```

带参数的构造方法

- **通过调用带参数的构造方法，简化对象初始化的代码**

```
public class Teacher6Test {
        public static void main(String[] args) {
                Teacher6  teacher =  new Teacher6(“Mary",
                                                  23, "本科", "咨询师");
                System.out.println(teacher.introduction());
        }
}
```

创建对象时，一并完成了对象成员的初始化工作

```
大家好！我是Mary,我今年23岁，学历本科，目前职位是咨询师
```

构造函数(Constructor)

- 构造函数实例化类对象的格式
  - 类名 对象名=new 构造函数（实际参数）
  - New关键字的作用
    ① 为对象分配内存空间。
    ② 引起对象构造方法的调用。
    ③ 为对象返回一个引用。

不带参数构造方法（默认构造方法）

- Java语言中，每个类都至少有一个构造方法；
- 如果类的定义者没有显式的定义任何构造方法，系统将自动提供一个默认的构造方法：
  - 默认构造方法没有参数
  - 默认构造方法没有方法体
  - 默认的构造方法：Animal(){}
  - 所以：不编写构造方法就能用new Xxx()创建类的实例。

不带参数构造方法（默认构造方法）

- 无参构造函数创建对象时，成员变量的值被赋予了数据类型的隐含初值。
  - 例如：整型是 0，实型是 0.0，复合类型是 null 。

变量类型	默认值	变量类型	默认值
byte	0	short	0
int	0	long	0L
float	0.0f	double	0.0d
char	'\u0000'	boolean	false
引用类型	null		

不带参数构造方法（默认构造方法）

- 在Java里，如果一个类没有明显的表明哪一个类是它的父类，Object类就是它的父类。
- 如果类中没有定义构造函数，编译器会自动创建一个默认的不带参数的构造函数。
- 如果程序员为类定义了构造函数，Java就不会为该类创建默认的不带参数的构造函数。
- 注意：如果类中提供的构造函数都不是无参数构造函数，却企图调用无参数构造函数初始化此类的对象，编译时会产生语法错误

构造方法重载

- 通过调用不同的构造方法来表达对象的多种初始化行为
- 例如：
  - 默认情况下，教师来自北京中心，初始化时，只需提供教员姓名
  - 有时，需要提供所在中心名称及教员姓名

构造方法重载是方法重载的典型示例

构造方法重载

```
public class Teacher7 {
        private String name; // 教员姓名
        private String school = "北京中心";  // 所在中心

        public Teacher7(String name) {
                this.name = name;  //设定教员姓名
        }
        public Teacher7(String name, String school) {
                this.name = name;     //设定教员姓名
                this.school = school; //设定教员的所在中心
        }
        public String introduction() {
                return "大家好！ 我是" + school + "的" + name ;
        }
}
```

通过调用不同的构造方法来表达对象的多种初始化行为

```
public class Teacher7Test {
    public static void main(String[ ] args) {
        Teacher7  teacher1 = new Teacher7(“Mary");
        System.out.println(teacher1.introduction());

        Teacher7  teacher2 = new ATeacher7(“Jack”, "天津中心");
        System.out.println(teacher2.introduction());
    }
}
```

```
大家好！我是北京中心的Mary
大家好！我是天津中心的Jack
```

回答问题

- 给定如下Java代码，请指出代码中的错误，并解释原因。

回答问题

```
public class Sample {
    private int x;
√   public Sample() {
        x = 1;
    }
√   public Sample(int i) {
        x = i;
    }
√   public int Sample(int i) {
        x = i;
        return x++;
    }
√   private Sample(int i, String s){}

√   public  Sample(String s,int i){}

×   private Sampla(int i){
        x=i++;
    }
√   private void Sampla(int i){
        x=i++;
    }
}
```

无参构造方法

带参构造方法

不是构造方法

带参构造方法

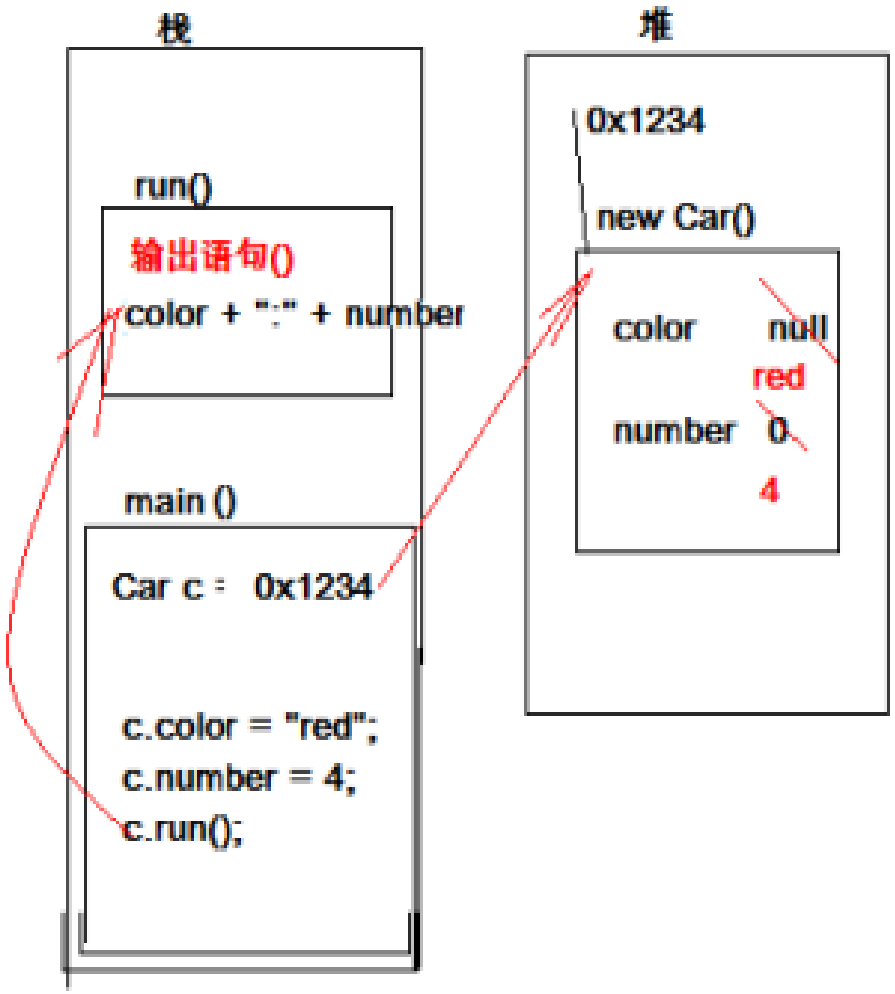
带参构造方法

名称与类名不相同

不是构造方法

对象的内存图解

```
public class Car {
    String color;
    int number;

    void run() {
        System.out.println(color + ":" + number);
    }
}

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

        Car c = new Car();

        c.color = "red";
        c.number = 4;
        c.run();

    }
}
```



内存管理

- 内存管理通常有两种方法：
  ① 一种方法是由程序员在编写程序时显式地释放内存，例如C++。
  ② 另一种方法是由语言的运行机制自动完成，例如 Smalltalk，EiffeI和 Java。

JAVA内存管理（垃圾自动回收机制）

- Java虚拟机后台线程负责内存的回收。
- 垃圾强制回收机制：Java系统提供了方法“**System.gc()**”和“**Runtime.gc()**”方法来强制立即回收垃圾(但系统并不保证会立即进行垃圾回收)。
- 判断一个存储单元是否是垃圾的依据是：**该存储单元所对应的对象是否仍被程序所用。**

Java内存管理（续）

- 判断一个对象是否仍为程序所用的依据是：是否有引用指向该对象。
- Java的垃圾收集器自动扫描对象的动态内存区，对所引用的对象加标记，然后把没有引用的对象作为垃圾收集起来并释放出去。
- Java虚拟机可以自动判断并收集到“垃圾”，但一般不会立即释放它们的存储空间。
- Java系统自己定义了一套垃圾回收算法，用来提高垃圾回收的效率。

Java内存强制回收

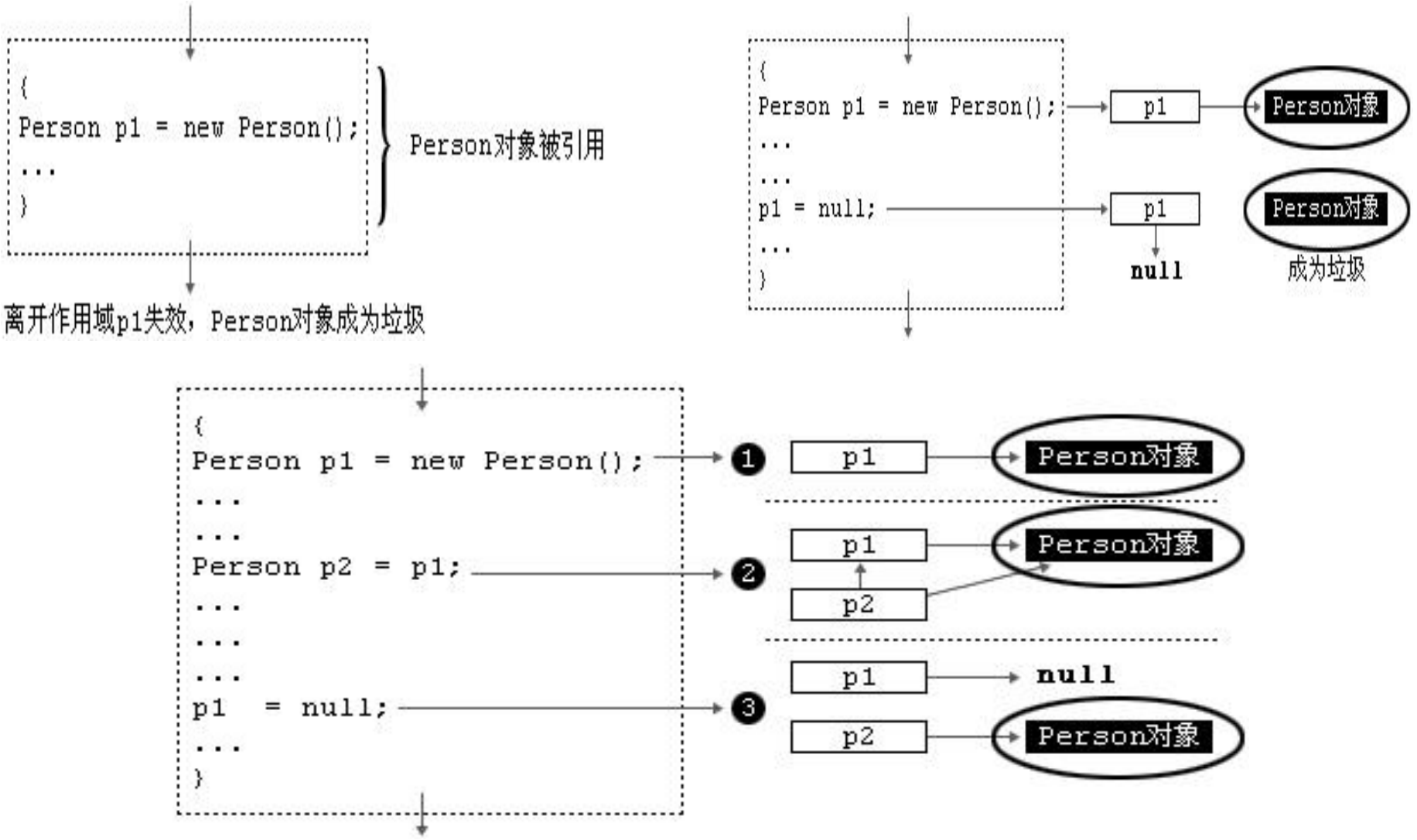
- **System.gc();**强制系统回收垃圾内存
- **Runtime.gc();**强制系统回收垃圾内存
- Java没有提供析构方法，但提供了一个类似的方法：protected void finalize()。
- Java虚拟机在回收对象存储单元之前先调用该对象的finalize方法，如果该对象没有定义finalize方法，则java系统先调用该对象默认的finalize方法。

J_Finalize.java

```
package com.buaa.classEx;

class J_Book extends Object {
    private String m_name;

    J_Book(String name) {
        m_name = name;
    }

    protected void finalize() {// ////覆盖
        System.out.println("Book,\"" + m_name + "\",is destroyed!");
    }
}

public class J_Finalize {
    public static void main(String args[]) {
        J_Book book1 = new J_Book("Gone with Wind");// ///
        new J_Book("Java How To Program"); // drop the reference
        new J_Book("Roman Holiday");
        System.gc();// ////
        book1 = new J_Book("thinking in Java");
    }
}
```

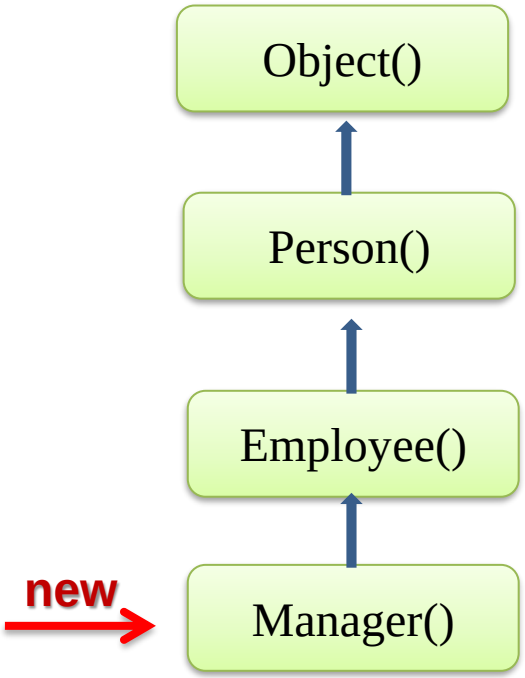
```
Book,"Roman Holiday",is destroyed!
Book,"Java How To Program",is destroyed!
```

对象的生命周期


{
Person p1 = new Person();
...
}
Person对象被引用
离开作用域p1失效，Person对象成为垃圾
{
Person p1 = new Person();
...
...
p1 = null;
...
}
p1
Person对象
p1
null
Person对象
成为垃圾
{
Person p1 = new Person();
...
...
Person p2 = p1;
...
...
...
p1 = null;
...
}
1
p1
Person对象
2
p1
Person对象
p2
3
p1
null
p2
Person对象

小结（1）

- **对象**是程序所处理数据的最主要的载体，数据以实例变量的形式存放在对象中。每个对象在生命周期的开始阶段，Java虚拟机都需要为它**分配内存**，然后对它的实例变量进行初始化。
- **对象构造顺序：**用new语句创建类的对象时，Java虚拟机会从最上层的父类开始，依次执行各个父类以及当前类的构造方法，从而保证来自于对象本身以及从父类中继承的实例变量都被正确地初始化。



小结（2）

- **对象构造顺序**：当子类的构造方法没有通过super语句调用父类的构造方法，那么Java虚拟机会自动先调用父类的默认构造方法。

- **匿名对象**：当一个对象不被程序的任何引用变量引用，对象就变成无用对象，它占用的内存就可以被垃圾回收器回收。

小结（3）

- **合理地使用内存**：每个对象都会占用一定的内存，而内存是有限的资源，为了合理地利用内存，在决定对象的生命周期时，应该遵循以下原则：
  - 当程序不需要再使用一个对象，应该及时清除对这个对象的引用，使它的内存可以被回收。
  - **重用已经存在的对象**。程序可通过类的静态工厂方法来获得已经存在的对象，而不是通过new语句来创建新的对象。（Java反射）

主要内容

- 类的设计原则
- Java程序的基本结构
- 类的定义
  - 成员变量的定义
  - 成员方法的定义
- 对象的生成、使用和清除
  - 默认构造函数（不带参数构造函数）
  - 带参数构造函数
  - 垃圾内存自动回收机制
- **匿名对象**
- 进一步理解引用类型变量
- 类的静态属性和静态方法（重点、难点）

匿名对象概念和使用

- 我们也可以不定义对象的句柄，而直接调用这个对象的方法。这样的对象叫做匿名对象
- 例如：
  - Student stu = new Student();//stu是对象，名字是stu
  - new Student();//这个也是一个对象，但是没有名字，称为匿名对象
  - new Student().show(); 匿名对象方法调用
- 匿名对象用完了之后就变成垃圾了，因为这个对象没有被栈内存中的变量指向，所有会被gc回收

匿名对象概念和使用

- 如果对一个对象只需要进行一次方法调用，那么就可以使用匿名对象。

- **我们经常将匿名对象作为实参传递给一个函数调用。**

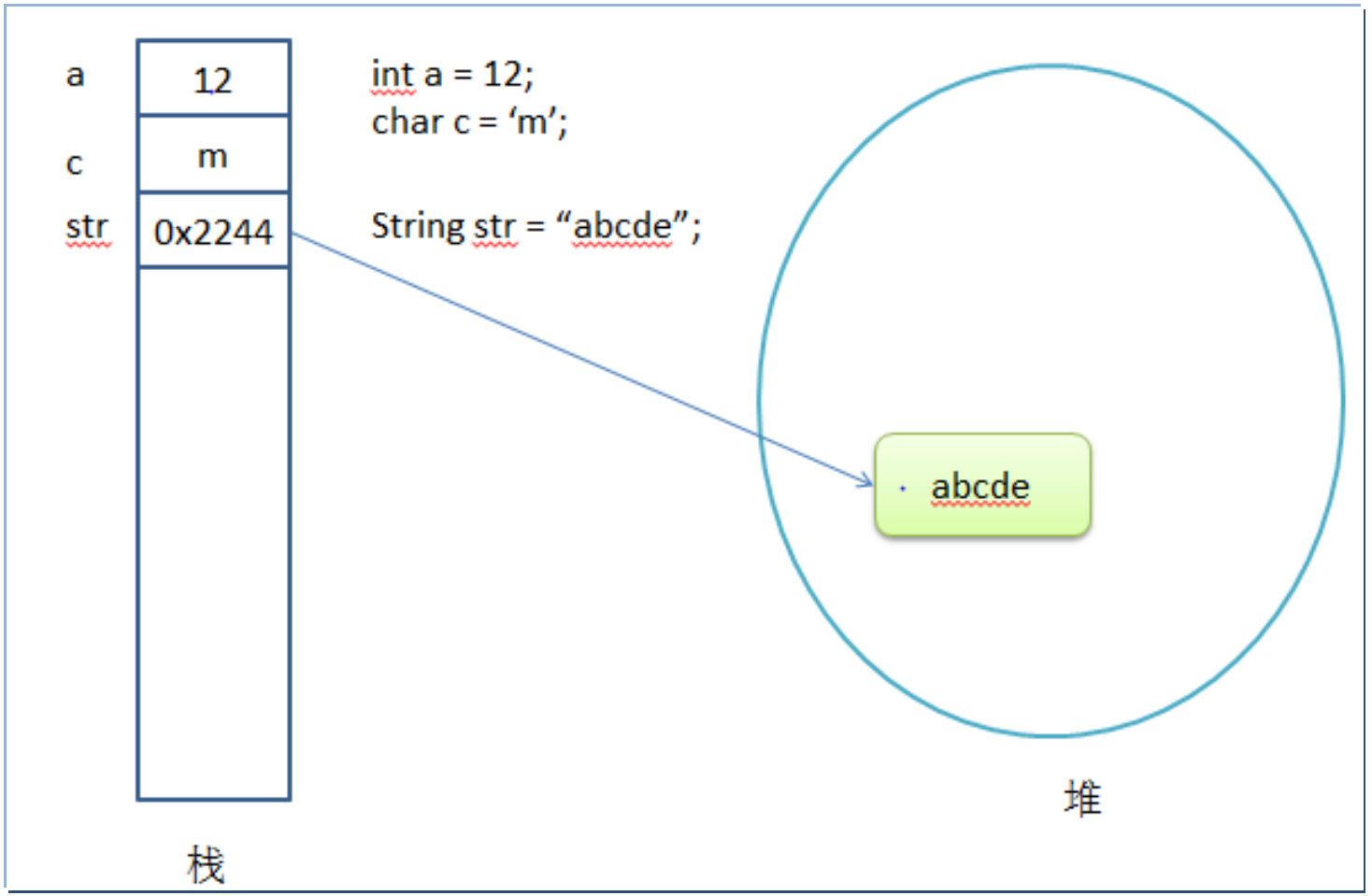
主要内容

- 类的设计原则
- Java程序的基本结构
- 类的定义
  - 成员变量的定义
  - 成员方法的定义
- 对象的生成、使用和清除
  - 默认构造函数（不带参数构造函数）
  - 带参数构造函数
  - 垃圾内存自动回收机制
- 匿名对象
- **进一步理解引用类型变量**
- 类的静态属性和静态方法

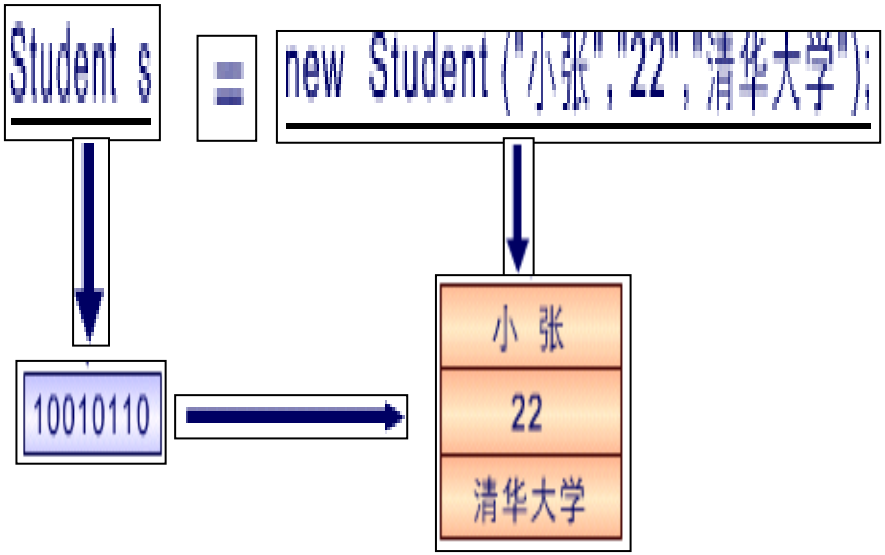
进一步理解引用类型变量

- 对象由两部分组成：
  - 对象的实体
  - 对象的引用

简单数据类型和复合数据类型内存分配

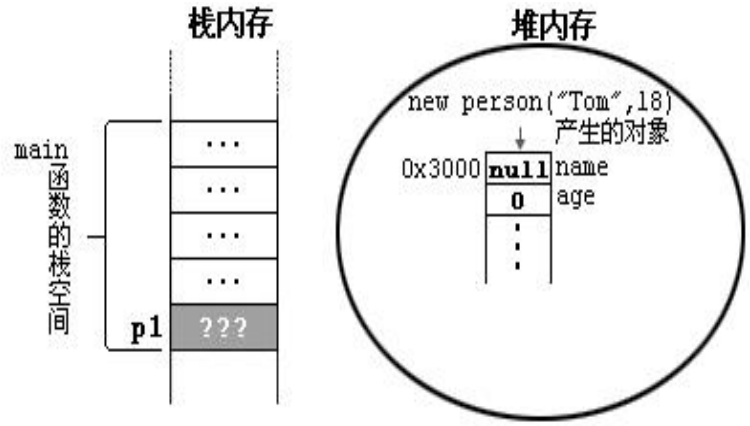


类是引用类型变量

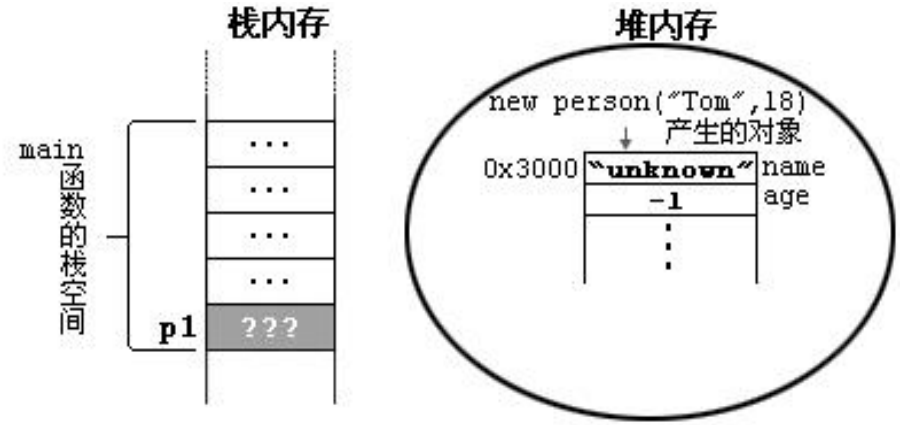


声明并不为对象分配内存空间，而只是分配一个引用空间；对象的引用类似于指针，是32位的地址空间，它的值指向一个中间的数据结构，它存储有关数据类型的信息以及当前对象所在的堆的地址，而对于对象所在的实际的内存地址是不可操作的，这就保证了安全性。

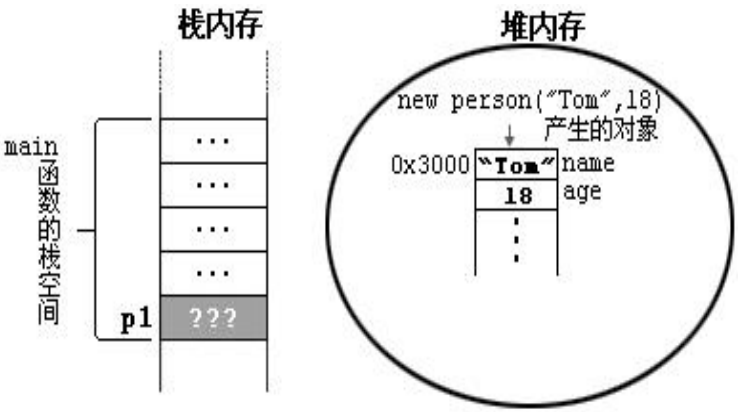
Person p1 = new Person(“Tom”,18) 的内存状态变化过程分析



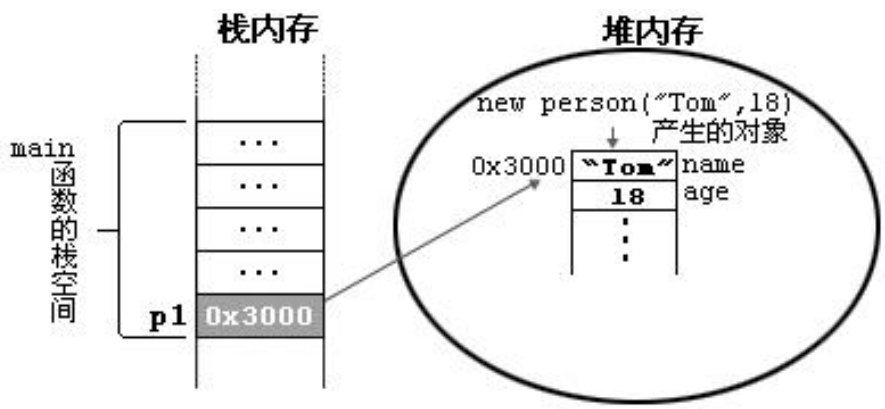
(2)



(1)



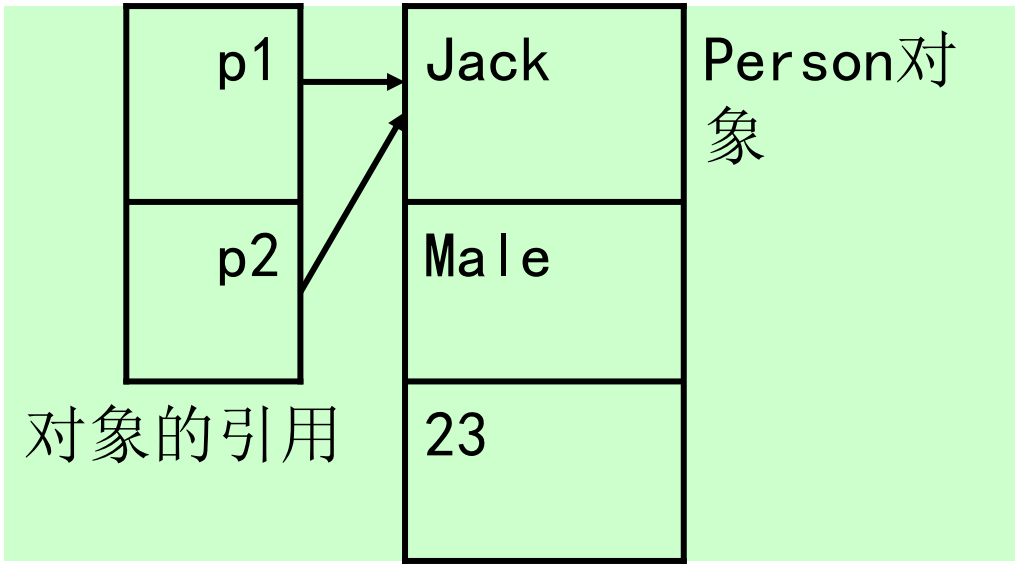
(3)



(4)

进一步理解引用类型变量

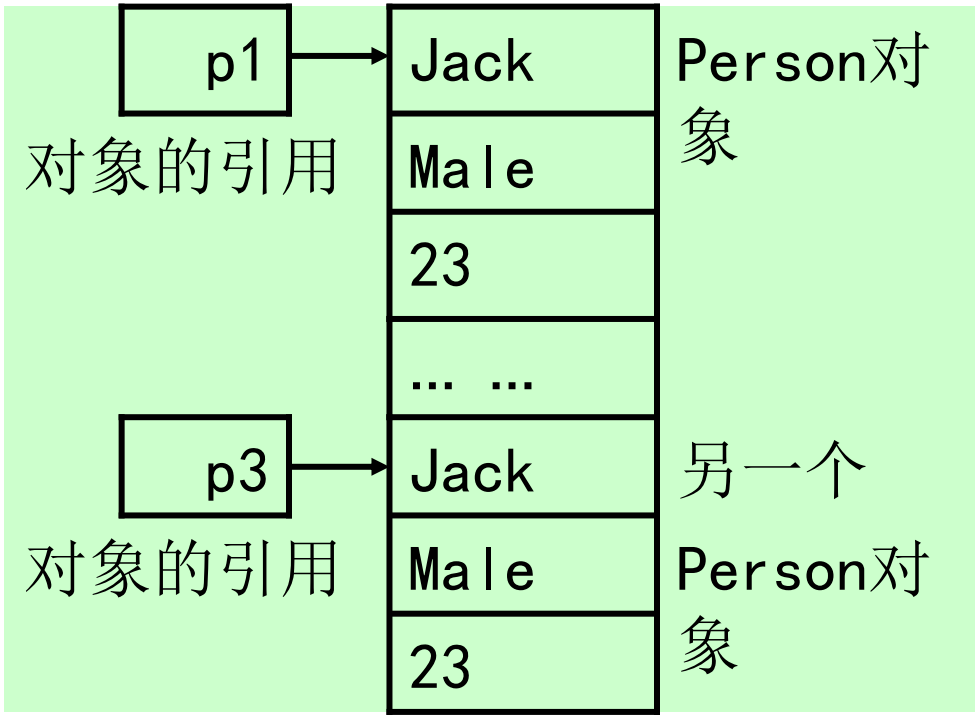
- Person p1=new Person(“Jack”,”Male”,23);
- Person p2=p1;



p1和p2指向同一个对象

进一步理解引用类型变量

- Person p1=new Person(“Jack”,”Male”,23);
- Person p3=new Person(“Jack”,”Male”,23);



P1和P3指向不同的对象

进一步理解引用类型变量

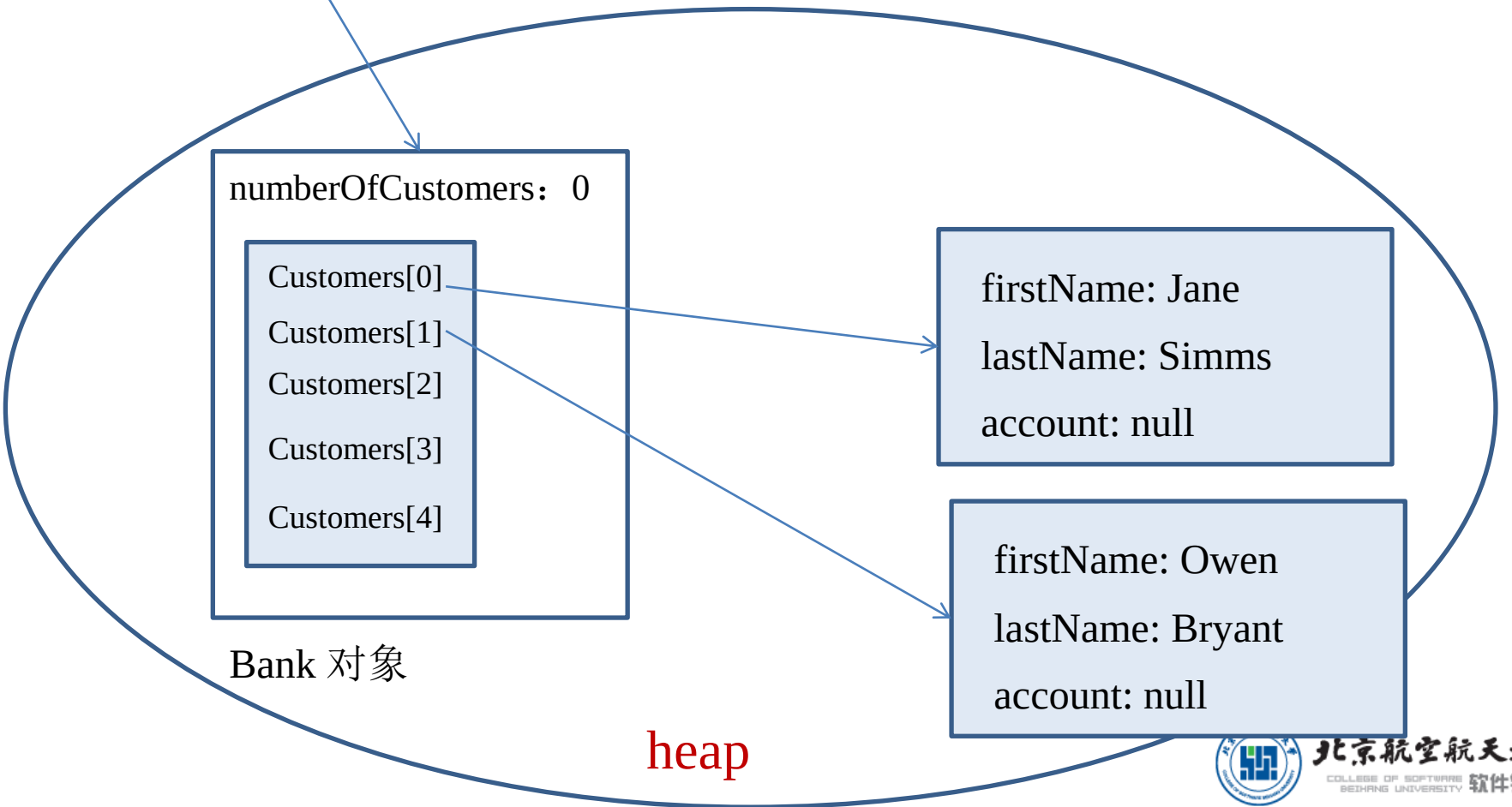
```
public Bank() {
    customers = new Customer[5];
}
```

```
public void addCustomer(String f, String l){
    Customer customer = new Customer(f, l);
    customers[numberOfCustomers] = customer;
    numberOfCustomers++;
}
```

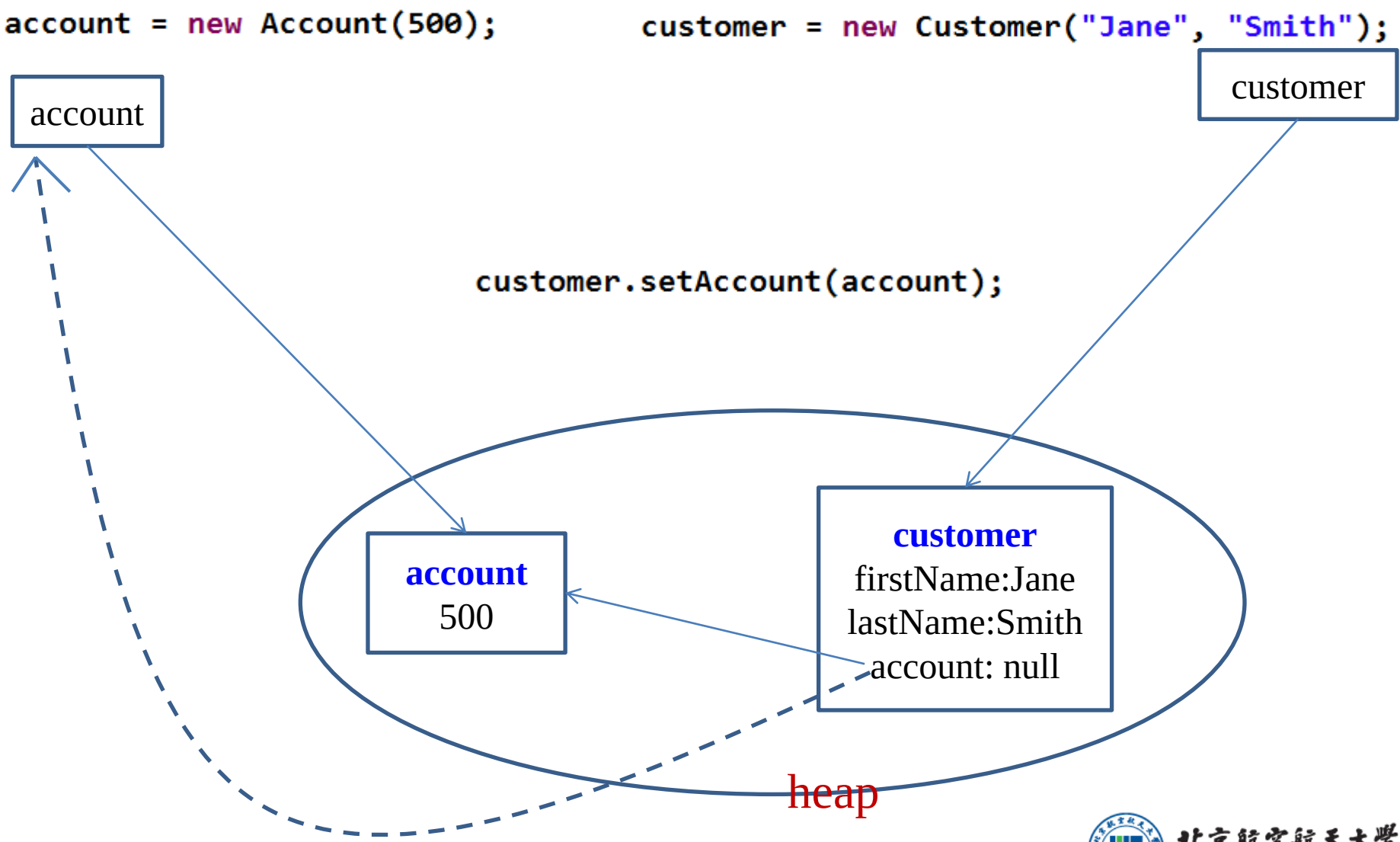
进一步理解引用类型变量

```
Bank      bank = new Bank();
```

```
bank.addCustomer("Owen", "Bryant");
bank.addCustomer("Jane", "Simms");
```



进一步理解引用类型变量



注意事项

- 自动初始化只用于成员变量，方法体中的局部变量不能被自动初始化，必须赋值后才能使用。

Person. java(分清楚全局变量和局部变量)

```
package com.buaa.classEx;
import java.io.*;
public class Person {
    private String name;
    private String sex;
    private int age;
    public Person() {
        int i=0;
        System.out.println("父类构造函数");
    }
    public Person(String n, String s, int a) {
        name = n;
        sex = s;
        age = a;
    }
    public void display() {
        System.out.println("姓名：" + name + "," + "性别：" + sex + "," + "年龄：" + age);
    }
```

方法体中的局部变量不能被自动初始化，必须赋值后才能使用。

Person. java

```
    public static void main(String[] args) {
        Person p0=new Person();////
        p0.display();////
        Person p = new Person("张三", "男", 20);
        p.display();
        int i=0;///局部变量必须初始化后才能使用
        System.out.println(i);
        System.out.println(p0.toString());///
    }


}
```

类的访问机制：

- 在一个类中的访问机制：类中的方法可以直接访问类中的成员变量

- 在不同类中的访问机制：先创建要访问类的对象，再用对象访问类中定义的成员。

局部变量和成员变量区别

- 区别一：定义的位置不同
  - 定义在类中的变量是成员变量
  - 定义在方法中或者{}语句里面的变量是局部变量
- 区别二：在内存中的位置不同
  - **成员变量存储在堆内存的对象中**
  - **局部变量存储在栈内存的方法中**

局部变量和成员变量区别

- 区别三：声明周期不同
  - 成员变量随着对象的出现而出现在堆中，随着对象的消失而从堆中消失
  - 局部变量随着方法的运行而出现在栈中，随着方法的弹栈而消失
- 区别四：初始化不同
  - **成员变量因为在堆内存中，所有成员变量具有默认的初始化值**
  - **局部变量没有默认的初始化值，必须手动的给其赋值才可以使用。**

toString()方法 （特殊点强调）

- 在java中，所有对象都有默认的toString()这个方法
- 创建类时没有定义toString()方法，**输出对象时会输出对象的哈希码值（对象的内存地址）**
- 它通常只是为了方便输出，比如System.out.println(xx)，（xx是对象），括号里面的”xx”如果不是String类型的话，就自动调用xx的toString()方法
- 它只是sun公司（Oracle）开发java时为了方便所有类的字符串操作而特意加入的一个方法

例如：

```
class Desk {
    private String color;
    private int length;
    private int width;
    private int height;

    public Desk() {
        super();
        // TODO Auto-generated constructor stub
    }

    public Desk(String color, int height, int length, int width) {
        super();
        // TODO Auto-generated constructor stub
        this.color = color;
        this.height = height;
        this.length = length;
        this.width = width;
    }
```

```
public static void main(String[] args){
    //test phrase 1
    Desk desk1=new Desk();
    System.out.println("desk1's volume is:"+desk1.volume());
    desk1.color="red";
    desk1.setColor("red");
    desk1.height=40;
    desk1.setHeight(40);
    desk1.length=20;
    desk1.setLength(20);
    desk1.width=20;
    desk1.setHeight(20);
    System.out.println("desk1's volume is:"+desk1.volume());

    //test phrase 2
    Desk desk2=new Desk("red", 30, 30, 30);
    System.out.println("desk2's volume is:"+desk2.volume());

    System.out.println(desk2.toString());
    System.out.println(desk2);
}
```

```
desk1's volume is:0
desk1's volume is:8000
desk2's volume is:27000
com.buaa.classEx.Desk@777d57d6
com.buaa.classEx.Desk@777d57d6
```

案例分析：

- 设计程序满足以下四点：

① 在不同迅腾国际培训中心（北京中心，杭州中心等），你会感受到相同的环境和教学氛围。

② Xtgj中心的默认值是“杭州中心”

③ 编写Xtgj中心的toString()，输出该中心的描述信息

④ 编写测试类

Xtgj类	
属性	中心全称 中心教室数目 中心机房数目
方法	展示中心信息

代码分析：

```
public class Xtgj{
    //定义迅腾国际中心的属性
    String schoolName="杭州中心";   // 中心的全称，默认为“杭州中心”
    int classNum;            //教室的数目
    int labNum;              //机房的数目

    // 定义迅腾国际中心的方法
    public String toString() {
        return schoolName + "培训学员"
            + "\n" + "配备："+ classNum + "教" + labNum + "机";
    }
}
```

成员变量

```
public String toString(){
    return 字符串；  //方法体
}
```

定义类的toString()方法，
用于输出类相关的信息

创建和使用对象示例

- 创建“北京中心”对象

```
public class InitialSchool {

    public static void main(String[ ] args){
        Xtgj center = new Xtgj();

        System.out.println(center);

        center.schoolName = "北京中心";
        center.classNum = 10;
        center.labNum = 10;

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

说一说看到什么效果？

```
杭州中心培训学员
配备：0教0机
北京中心培训学员
配备：10教10机
北京中心培训学员
配备：10教10机
```

这里使用center.toString()和center作用相同——更简便