[一、JAVA简介](#header-n3)  
 [二、JAVA快速入门](#header-n8)  
 [1.名词解释](#header-n16)  
 [三、面向对象程序设计](#header-n17)  
 [（一）封装性](#header-n29)  
 [1.类与对象](#header-n31)  
 [a.类（class）](#header-n33)  
 [b.类的构造方法](#header-n36)  
 [c.对象创建及其实例化](#header-n42)  
 [（二）继承性](#header-n63)  
 [1.基本概念](#header-n64)  
 [2.方法覆盖与方法重载](#header-n71)  
 [3.属性覆盖](#header-n75)  
 [4.抽象方法](#header-n94)  
 [5.接口](#header-n108)  
 [（三）多态性](#header-n127)  
 [1.上溯](#header-n129)  
 [2.向下转型](#header-n135)  
 [专题一：时间观](#header-n138)  
 [专题二：关于子类父类的那些破事](#header-n145)  
 [专题三：那些细碎不知道归纳到哪的玩意](#header-n156)  
 [1.基本类型](#header-n157)  
 [2.static 静态方法与属性](#header-n161)  
 [3.控制设计](#header-n170)  
 [4.Javadoc](#header-n174)  
 [5.作用域](#header-n177)

## 一、JAVA简介

Java是基于JVM虚拟机（高低版本完美兼容）的跨平台语言，可移植性极佳；介于编译型语言和解释型语言之间，将代码编译成一种“字节码”，它类似于抽象的CPU指令

<<<<<<< HEAD  
我是一个编剧，创造了这个java世界，而虚拟机类似于导演

=======

7f1e91df6c5133b1ebafbf6d795a2e939ef13d6d

## 二、JAVA快速入门

### 1.名词解释

## 三、面向对象程序设计

<<<<<<< HEAD  
**主谓结构**

=======

7f1e91df6c5133b1ebafbf6d795a2e939ef13d6d  
面向对象主要是关注对象的属性与方法，定义好相关对象，具体的对象正在做什么事情的流程，我们不予关注，唯一能做的只能控制时间与重置。“我们是创世者，但是七天后就得走，且不干预人间任何事情。” 面向过程语言如C语言，不存在对象概念，主要是控制目标实现的的每一个步骤，每一个步骤都是可操控的。

面向对象的程序设计有3个主要特征：封装性、继承性、多态性。学习主要是需要形成自己的程序时空观。

### （一）封装性

封装是为了有效描述对象，更简单的理解是将代码有效分隔，有利于程序员阅读与代码debug修改 。“高类聚，低耦合”，将对象的属性与方法封装在一起，定义为一个程序单位(class)，其访问权限可人为定义。

#### 1.类与对象

类是一种结构定义，而对象是类的载体，一个类可以创造多种对象，两者不可以分割。

##### a.类（class）

类是由属性和方法组成。属性为类中包含的一些变量的定义，而方法是该类的一些操作行为。例如：

/\*定义了蜜蜂的类\*/  
public class Bee00{  
 int id; //蜜蜂的唯一标识  
 int x,y; //蜜蜂现在所处的位置坐标(最小单位：像素)  
 int honey; //当前蜜蜂采集蜂蜜的数量(单位：mg)  
//以上三个变量为蜜蜂的属性  
//以下为定义的蜜蜂的显示当前采蜜数方法（注意：存在返回值）  
 public void showHoney()  
 {  
 System.out.println("Bee: "+id+" has ("+honey+"mg) honey ");  
 }

##### b.类的构造方法

在类中存在类的同名方法，叫做类的构造方法。只要是类，必须存在构造方法。用于类的生（初始化）

<u>注意：构造方法是没有返回值的，因为在创建对象，实例化对象时需要将返回对象的全部内容？？</u>

public class Bee00{  
 int id; //蜜蜂的唯一标识  
 int x,y; //蜜蜂现在所处的位置坐标(最小单位：像素)  
 int honey; //当前蜜蜂采集蜂蜜的数量(单位：mg)  
   
 public Bee00(int id,int x,int y)  
 {   
 this.id = id;//this调用的是当前类，这句意为传入参数的id赋值给这个类的id初值  
 this.x = x;  
 this.y = y;  
 this.honey = 0;  
 System.out.println("Bee: "+id+" come from ("+x+","+y+")");  
 }

**this关键字**

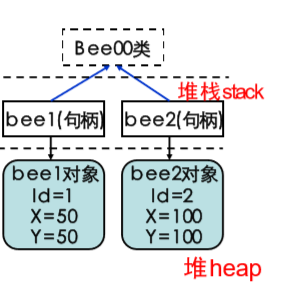
this表示当前对象，可以使用this调用本类的构造方法，具体见上。

##### c.对象创建及其实例化

通过new关键字对对象进行实例化

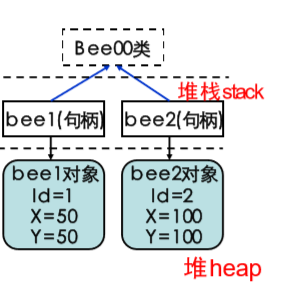
public class Bee00{  
 int id; //蜜蜂的唯一标识  
 int x,y; //蜜蜂现在所处的位置坐标(最小单位：像素)  
 int honey; //当前蜜蜂采集蜂蜜的数量(单位：mg)  
   
 public Bee00(int id,int x,int y)  
 {   
 this.id = id;  
 this.x = x;  
 this.y = y;  
 this.honey = 0;  
 System.out.println("Bee: "+id+" come from ("+x+","+y+")");  
 }  
   
 public static void main(String[] args)//实例化在class中实现，参数为字符串类型的数组  
 {  
 Bee00 bee1 = new Bee00(1,50,50);//在主程序中实例化对象，声明对象名为bee1，然后调用构造方法(传递参数进去)进行实例化  
 Bee00 bee2 = new Bee00(2,100,100);  
 }  
}

<<<<<<< HEAD  
其对象通过操作句柄操作，具体**内存映像** 为下图



我们可调用的只有句柄，对象是无法直接调用的，句柄对应于对象的声明名字。句柄存储在栈中，而对象的属性存储在堆中，而方法在堆栈之外的其他内存（代码区）【栈内存小，调用速度快；堆内存大，调用速度慢】

其对象通过操作句柄操作，具体内存映像为下图



我们可调用的只有句柄，对象是无法直接调用的，句柄对应于对象的声明名字。句柄存储在栈中，而对象的属性存储在堆中，而方法在堆栈之外的其他内存（栈内存小，调用速度快；堆内存大，调用速度慢）

7f1e91df6c5133b1ebafbf6d795a2e939ef13d6d

**调用方法：（属性）bee1.id bee2.X （方法）bee1.showhoney()**

当单纯打印句柄时，其返回值仅为句柄在栈中的地址。对象实例化称为对象的“生”，如果其句柄不再指向该对象对应的空间，这就称为对象的“死”。

### （二）继承性

#### 1.基本概念

继承的目的是为了条理清晰地代码重用 ，继承后子类拥有父类的结构、属性与方法。**值得注意：在实际内存中，父类并不存在，只存在子类的内存空间！！！**

**继承方法：**

class 父类{}  
class 子类 extends 父类{}  
  
class Bee{//父类：Bee,定义了采蜜容量，与采蜜的方法  
 int honeyBag = 5;  
 public void pickHoney(){  
 honeyBag ++;  
 }   
}  
  
public class Ext1 extends Bee{//public表示访问权限  
 public static void main(String[] args)  
 {   
 //Bee b = new Bee();对父类实例化  
 Ext1 e = new Ext1();//实际上一般只对子类实例化   
 e.pickHoney();//子类也拥有的父类的方法  
 e.pickHoney();  
 e.pickHoney();  
 System.out.println("Mikee bee has "+e.honeyBag+" kg honey");  
 }  
}

**访问权限修饰符：**

访问权限大小关系：private<default<public

如果父类为private，子类将无法继承；如果父类为public，子类也必须为public，不然无法进行编译。

#### 2.方法覆盖与方法重载

方法覆盖要考虑权限，子类权限不能小于父类权限。另外子类方法一旦覆盖父类，每次**在主程序**调用**子类句柄** 使用子类方法。但是在子类中仍可以使用super.父类方法名称();调用父类的方法。

class Bee{  
 int honeyBag = 0;  
 public void pickHoney(){  
 honeyBag ++;  
 }   
}  
  
public class Ext2 extends Bee{  
 public void pickHoney(){  
 honeyBag += 5;  
 }  
  
 public static void main(String[] args)  
 {  
 Ext2 e = new Ext2();  
 e.pickHoney();//此时调用采蜜方法，其为每次加5，子类覆盖了父类的方法  
 e.pickHoney();  
 e.pickHoney();  
 System.out.println("Mikee bee has "+b.honeyBag+" kg honey");  
 }  
}

方法重载即可使用同名方法，但是参数不同，虚拟机会根据传递参数的不同选用不同的方法进行调用。

#### 3.属性覆盖

如果子类声明了和父类同名的属性，子类属性不会覆盖父类属性，而是同时存在两类属性。调用this，即为调用子类属性；调用父类属性通过使用super关键字。

class Father{  
 int speed;  
 public Father(int speed){  
 this.speed = speed;  
 }  
  
 public void run(){  
 System.out.println("Father run "+speed+" km/h");  
 }  
  
}  
  
class Son extends Father{  
 int speed;  
 public Son(int speed){  
 super(speed-5);//设定父亲的speed，父亲比儿子速度慢5km/h  
 this.speed = speed;//设定自己的speed  
 }  
  
 public void run(){  
 //super.run();//打印父亲跑的信息(通过调用父类的方法)  
 System.out.println("Father run "+super.speed+" km/h");//通过调用父类的属性  
 System.out.println("Son run "+speed+" km/h");  
 }  
  
}  
  
public class Job2{  
 public static void main(String[] args) {  
 Son s = new Son(30);  
 s.run();  
 }  
}

**super关键字：**

和this一样，调用父类的构造方法时要放在子类构造方法的第一句 因为子类不会继承父类的构造方法，子类默认的构造方法是编译器自动生成的，不是继承的。如果没有写，编译器会在子类构造方法第一行加上super（）；如果父类有参数，则编译不会通过。因此需先调用父类的构造方法，使用**super（参数）；**

|  |  |
| --- | --- |
| 使用super情景 | 调用方法 |
| 在子类构造方法中调用父类构造方法 | super(父类构造方法参数)； |
| 在子类中使用父类属性 | super.父类属性名称 |
| 在子类中调用父类的方法 | super.父类方法名称(参数)； |

#### 4.抽象方法

抽象方法只需声明(使用abstract关键字），而不需要实现。（一个子类只能继承一个抽象方法）抽象类必须被子类继承，且子类必须复写抽象类中的全部抽象方法。**注：抽象方法不能使用private声明，否则子类无法覆盖**

**可以包含非抽象方法！！！** （适用于很多方法需要实现，父类会统统帮你实现）

abstract class 抽象类名称{  
 属性;  
 访问权限 返回值类型 方法名称(参数){//普通方法  
 return 返回值;  
 }  
 访问权限 abstract 返回值类型 方法名称(参数); //抽象方法，没有方法体  
}  
  
abstract class Bee{  
 int honeyBag = 0;  
 public abstract void pickHoney();//抽象方法  
}  
  
class Honee extends Bee{  
 public void pickHoney()  
 {  
 honeyBag +=1;  
 }  
}  
  
class Mikee extends Bee{  
 public void pickHoney()  
 {  
 honeyBag +=5;  
 }  
}  
  
public class Ext4{  
 public static void main(String[] args)  
 {  
 Honee h = new Honee();  
 Mikee m = new Mikee();  
 h.pickHoney();  
 m.pickHoney();  
 System.out.println("Honee has "+h.honeyBag+"kg and Mikee has "+m.honeyBag+"kg");  
 }  
}

#### 5.接口

接口是一种特殊的类，里面全部是由全局常量和公共的抽象方法所组成。**不允许有非抽象方法出现！！！** 接口中的访问权限无论写还是不写，都是public。接口就像干爹，可以认很多个，而父类只能认一个。 （适用于方法比较少，构造方法之间差别比较大的）

interface 接口名称{  
 全局变量;  
 抽象方法;  
}  
class 子类 implements 接口A,接口B{  
   
}  
  
public interface Runable  
{  
 public void run();  
}  
class People implements Runable{  
 public void run(){  
 System.out.println("People run……");  
 }  
}  
  
class Dog implements Runable{  
 public void run(){  
 System.out.println("Dog run……");  
 }  
}  
  
public class Interf1{  
 public void run(Runable r){  
 r.run();  
 }  
  
 public static void main(String[] args)  
 {  
 People p = new People();  
 Dog d = new Dog();  
 p.run();  
 d.run();  
 Interf1 f = new Interf1();  
 f.run(d);  
 }  
}

**final关键字**

• final修饰的类，不能被继承

• final修饰的属性，是常量 （放置在常量池中，Java里常量的名称有大写字母和\_组成）

• final修饰的方法，不能被覆盖（final关键字断子绝孙）

image-20201010104032290

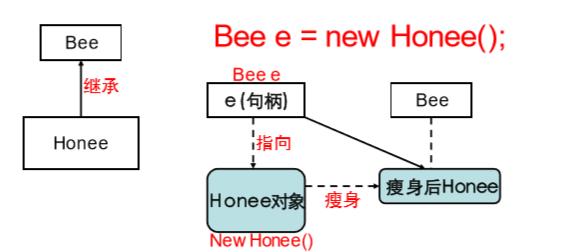
image-20201010104032290

### （三）多态性

java在面向对象中有两个主要体现：方法的重载与覆写、对象的多态性。多态指同一名称的方法或属性，由于所属对象不同， 而表现出的不同效果。

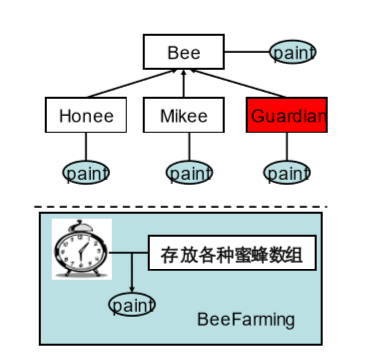
#### 1.上溯

指将子类的对象赋值给父类的句柄



因为父类句柄可调用的方法，子类都有。

abstract class Bee{  
 public abstract void paint();  
}  
  
class Honee extends Bee  
{  
 public void paint()  
 {  
 System.out.println("画一只小蜜蜂");  
 }  
}  
  
class Mikee extends Bee  
{  
 public void paint()  
 {  
 System.out.println("画一只大蜜蜂");  
 }  
}  
  
class Guardian extends Bee  
{  
 public void paint()  
 {  
 System.out.println("画一只守卫蜜蜂");  
 }  
}  
  
public class PaintAllBee{  
 Bee[] bees;  
   
 public PaintAllBee(){  
 bees = new Bee[3];  
 }  
   
 public Bee[] getAll(){  
 return bees;  
 }  
   
 public void paintAll()  
 {  
 for(int i=0;i<3;i++)  
 if(bees[i]!=null)  
 bees[i].paint();  
 }  
 public static void main(String[] args)  
 {  
 PaintAllBee pab = new PaintAllBee();  
 Bee[] x = pab.getAll();//通过这个父类句柄的数组实现整体的调度  
 x[0] = new Honee();  
 x[1] = new Mikee();  
 x[2] = new Guardian();  
 pab.paintAll();  
 }  
}  
>>>画一只小蜜蜂  
 画一只大蜜蜂  
 画一只守卫蜜蜂



#### 2.向下转型

向下转型一定要先向上转型

## 专题一：时间观

类中的类属性与类方法（只能一次）--->

父类的属性--->父类的构造方法(对属性的最终初始化）--->父类的普通方法--->

子类的属性--->子类的构造方法--->子类的普通方法

**注意：子类构造方法初始化时会隐性地对父类构造方法的无参数构造super（），如果父类是个含参构造方法，则会报错传递参数不正确，这时需要手动调用super（参数）**

public class StaticVar{  
 static int x1 = prt("static variable initialized");//1.静态属性初始化  
 final int x3 = prt("final variable initialized"); //4.非静态属性初始化  
 static final int x4 = prt("final static variable initialized");//2.静态属性初始化，final没有特别的初始化特征  
 int x2 = prt("variable initialized");//5.非静态属性初始化  
 static int prt() {   
 System.out.println("static fangfa initialized");  
   
 }  
 static void t()  
 {  
 int i = prt("static method variable initialized");？？？？  
 }  
   
 public static void main(String[] args)//3.静态方法初始化  
 {  
 System.out.println("before StaticVar initial");  
 StaticVar st = new StaticVar();   
 }  
}  
>>>  
static variable initialized  
final static variable initialized  
before StaticVar initial  
final variable initialized  
variable initialized

## 专题二：关于子类父类的那些破事

1.在不存在子类方法覆盖时，子类对象调用的从父类继承的方法不能操控子类的属性；除非子属性也是继承来的，直接上溯到操作父类

class Bee{  
 int honeyBag = 0;  
 public void pickHoney(){  
 honeyBag ++;  
 }  
}  
  
public class Test extends Bee{  
 int honeyBag = 2;  
  
 public static void main(String[] args) {  
 Test b = new Test();//子类对象赋给子类句柄  
 b.pickHoney();//父类对象操作子类属性失败  
 System.out.println(b.honeyBag);//打印子类属性  
 }  
}  
>>>2

class Bee{  
 int honeyBag = 2;  
 public void pickHoney(){  
 honeyBag ++;  
 }  
  
  
public class Test extends Bee{  
 int honeyBag = 4;  
 public void pickHoney(){  
 honeyBag += 5;  
}  
  
 public static void main(String[] args) {  
 Test b = new Test();//子类对象赋给子类句柄  
 b.pickHoney();//调用子类方法，操作子类对象成功  
 System.out.println(b.honeyBag);//输出子类属性  
 }  
}  
>>>9

不存在方法覆盖时，子类对象赋值给父类句柄后，？？？？？

class Bee{  
 int honeyBag = 2;  
 public void pickHoney(){  
 honeyBag ++;  
 }  
}  
  
public class Test extends Bee{  
 public static void main(String[] args) {  
 Bee b = new Test();  
 b.pickHoney();  
 System.out.println(b.honeyBag);  
 }  
}  
>>>3

2.在**存在方法覆盖时，对象决定调用哪个方法，句柄判断访问哪个属性**

class Bee{  
 int honeyBag = 2;  
 public void pickHoney(){  
 honeyBag ++;  
 }  
}  
  
public class Test extends Bee{  
 int honeyBag = 4;  
 public void pickHoney(){  
 honeyBag += 5;  
}  
  
 public static void main(String[] args) {  
 Bee b = new Test();//将子类对象赋给父类句柄  
 b.pickHoney();//对象判断方法，调用子类方法  
 System.out.println(b.honeyBag);//句柄判断属性，打印父类属性  
 }  
}  
>>>2 打印的是没有操作过得父类属性

子类对象可以赋给父类句柄，因为父类句柄可调用方法属性，子类全部都继承到了，一定存在。但是父类不存在的方法父类句柄是无法调用的

**注意：父类对象无法赋给子类句柄，因为子类句柄中可能含有许多父类中没有的属性或者方法，可能会出现b.xx错误的情况**

class Bee{  
 int honeyBag = 2;  
 public void pickHoney(){  
 honeyBag ++;  
 }  
}  
  
public class Test extends Bee{  
 int honeyBag = 4;  
 public void pickHoney(){  
 honeyBag += 5;  
}  
  
 public static void main(String[] args) {  
 Bee b = new Bee();//父类对象赋给父类句柄  
 b.pickHoney();//调用父类方法操作父类属性  
 System.out.println(b.honeyBag);//打印父类属性  
 }  
}  
>>>3

## 专题三：那些细碎不知道归纳到哪的玩意

#### 1.基本类型

比如一些数据类型或者操作符，如int i=5;它不依赖对象，全部存储在栈中（堆中不存在，也是为了好方便调用嘛，直接变成句柄了），均有缺省值。

**特别的使用：可以通过查看API文档，将数据类型定义的变量包装成特定类的对象，从而调用类中的其他方法，甚至都不需要import java.lang**

public class test0 {  
   
 public static void main (String[] args){  
 Integer i = Integer.valueOf(13);  
 System.out.println(i);  
 System.out.println(i.toString());  
   
 }  
}  
>>>  
 13 //数字13  
 13 //字符串13

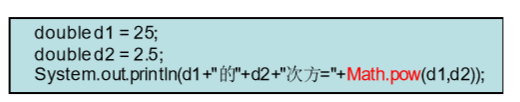
#### 2.static 静态方法与属性

静态方法与属性称为类方法与类属性（全局属性），他们同样不依赖对象，可以直接被调用，可以理解为共享对象空间。

类属性被放置在代码区（非堆栈），生命周期与这个世界相同

他们在程序编译时是最先被定义的，所以**静态方法中不要访问非静态属性（人家可能还没出生），但是可以访问静态属性** **但是只能被初始化一次**

正常代码最好不要出现全局属性，影响太大了



**关于main方法为什么在public类中**：

其实是java的倔强，它要求什么都在包里面，要做到面向对象，但是其实main方法与所在类（这个类名要求与文件名一致）屁关系都没有，因此main不能调用该类的属性，只能调用main自己的属性，毕竟main（）与日同辉，类属性（就算是静态也不要调用，显得自己没有水平）还没出生

#### 3.控制设计

缺省表示包内友好（包内均可访问），但是包外不能访问

protected:包内可以看到，包外继承后的子类可以看到自己属性

属性尽量使用private，方法使用public（可以公开使用）

#### 4.Javadoc

(记得复习python字符文档部分)

进入文件夹，然后使用 javadoc \*.java 可以生成全部类的javadoc API文件

#### 5.作用域