今日内容介绍
1、继承
2、抽象类
3、综合案例---员工类系列定义

01继承的概述

```
*A:继承的概念
    *a:继承描述的是事物之间的所属关系，通过继承可以使多种事物之间形成一种关系体系
    *b:在Java中，类的继承是指在一个现有类的基础上去构建一个新的类，
        构建出来的新类被称作子类，现有类被称作父类
 *B:继承关系的子类特点
    *a:子类会自动拥有父类所有非private修饰的属性和方法
```

02继承的定义格式和使用

```
*A:继承的格式
   class 子类 extends 父类 {}
*B:雇员(Employee)与研发部员工(Developer)案例:
   *cn.itcast.demo01包下:
   *Employee.java:
  /*
  * 定义员工类Employee
  */
 class Employee {
  String name; // 定义name属性

  public void work() {// 定义员工的工作方法
   System.out.println("尽心尽力地工作");
  }
 }
   *Developer.java:
    /*
  * 定义研发部员工类Developer 继承 员工类Employee
  * 继承了父类中所有非private修饰的成员变量
  */
 class Developer extends Employee {
  // 定义一个打印name的方法
  public void printName() {
   System.out.println("name=" + name);
  }
 }
   *测试员工类与研发部员工类:
       /*
     * 定义测试类
      */
 public class Example01 {
  public static void main(String[] args) {
   Developer d = new Developer(); // 创建一个研发部员工类对象
   d.name = "小明"; // 为该员工类的name属性进行赋值
   d.printName(); // 调用该员工的printName()方法
   d.work(); // 调用Developer类继承来的work()方法
  }
 }
   *通过子类对象既可以调用自身的非private修饰的成员,也可以调用父类的非private修饰的成
```

03继承的好处

```
*A:继承的好处：
    *1、继承的出现提高了代码的复用性，提高软件开发效率。
    *2、继承的出现让类与类之间产生了关系，提供了多态的前提。
```

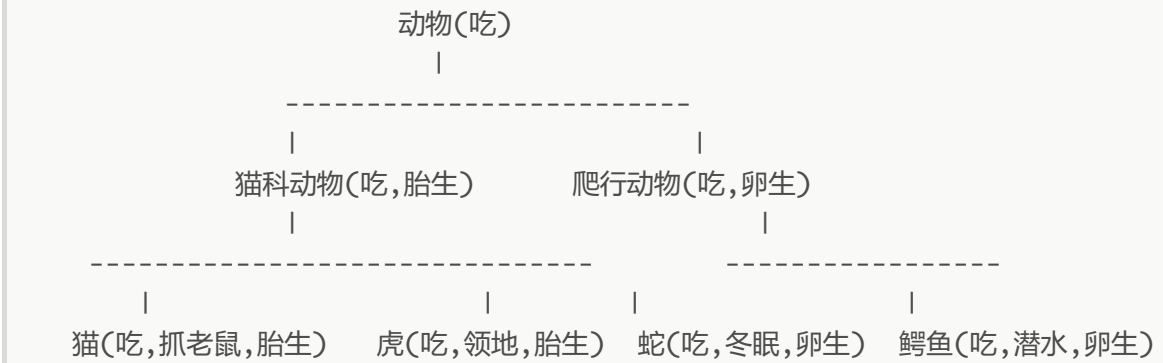
04继承的注意事项

```
 *A:继承的注意事项
  *a:在Java中，类只支持单继承，不允许多继承，也就是说一个类只能有一个直接父类，例如下面
      class A{}
      class B{}
      class C extends A,B{}  // C类不可以同时继承A类和B类
   假如支持多继承例如:
         class A{
          int a=3;
            public void method(){

            }
         }
      class B{
       int a=5;
       public void method(){

       }
      }
      class C extends A,B{

      }
      class Demo{
       public static void main(String[] args){
        C c=new C();
        System.out.println(c.a);//到底是调用A的还是B的成员变量??无法确定
        c.method();//到底是调用A的还是B的成员方法??无法确定
       }
      }

      *b:多个类可以继承一个父类，例如下面这种情况是允许的(就像你爹可以多个儿子,但是这些
      class A{}
      class B extends A{}
      class C extends A{}   // 类B和类C都可以继承类A

     *c:在Java中，多层继承是可以的，
        即一个类的父类可以再去继承另外的父类，
        例如C类继承自B类，而B类又可以去继承A类，这时，C类也可称作A类的子类。下面这种情
      class A{}
      class B extends A{}   // 类B继承类A，类B是类A的子类
      class C extends B{}   // 类C继承类B，类C是类B的子类，同时也是类A的子类
```

*d:在Java中，子类和父类是一种相对概念，
也就是说一个类是某个类父类的同时，也可以是另一个类的子类。
例如上面的这种情况中，B类是A类的子类，同时又是C类的父类。

05继承的体系.avi 11:00

*A:继承的体系:



*a:动物体系是对每个具体事物共性的抽取,子类的共性抽取形成父类
*b:父类:具有所有子类的共性内容
子类:不但有共性还有自身特有的内容
*c:整个继承体系,越向上越抽象,越向下越具体

06继承后子类父类成员变量的特点

```
A:继承后子类父类成员变量的特点
  a:子类的对象调用成员变量的时候,子类自己有,使用子类,子类自己没有调用的父类
   class Fu{
  //Fu中的成员变量。
  int num = 5;
 }

 class Zi extends Fu{
  //Zi中的成员变量
  int num2 = 6;
  //Zi中的成员方法
  public void show()
  {
   //访问父类中的num
   System.out.println("Fu num="+num);
   //访问子类中的num2
   System.out.println("Zi num2="+num2);
  }
 }

 class Demo{
  public static void main(String[] args)
  {
   Zi z = new Zi(); //创建子类对象
   z.show(); //调用子类中的show方法
  }
 }
b:当子父类中出现了同名成员变量
class Fu{
 //Fu中的成员变量。
 int num = 5;
}
```

```
class Zi extends Fu{
 //Zi中的成员变量
 int num = 6;
 void show(){
  //子类的局部变量
  int num=7

  //直接访问,遵循就近查找原则
           System.out.println(num);//7

  //子父类中出现了同名的成员变量时
  //在子类中需要访问父类中非私有成员变量时，需要使用super关键字
  //访问父类中的num
  System.out.println("Fu num="+super.num);//5


  //访问子类中的num2
  System.out.println("Zi num2="+this.num);//6
 }
}

class Demo5 {
 public static void main(String[] args)
 {
  Zi z = new Zi(); //创建子类对象
  z.show(); //调用子类中的show方法
 }
}
```

07继承后子类父类成员方法的特性_子类重写父类方法

```
A:继承后子类父类成员方法的特性
  a:子类的对象调用方法的时候,子类自己有,使用子类,子类自己没有调用的父类
    class Fu{
  public void show(){
   System.out.println("Fu类中的show方法执行");
  }
 }
 class Zi extends Fu{
  public void show2(){
   System.out.println("Zi类中的show2方法执行");
  }
 }
 public  class Test{
  public static void main(String[] args) {
   Zi z = new Zi();
   z.show(); //子类中没有show方法，但是可以找到父类方法去执行
   z.show2();
  }
 }
```

```
b:为什么要有重写?
    class Fu{
     public void method(){
      //上千行代码
            //Fu类中的方法最先存在,那么如果项目需求变了,该方法
            //功能不能够满足我们的需求,此时我们也不会去改这个方法
            //因为项目中可能有大量的功能已经使用到该方法,如果随意修改可能使调用该方法的
            //所以使用重写方式基于原有功能提供更强的功能
     }
    }
    class Zi extends Fu{

    }
c:子类中出现与父类一模一样的方法时，会出现覆盖操作，也称为override重写、复写或者覆盖
  class Fu{
 public void show(){
  System.out.println("Fu show");
 }
  }

  class Zi extends Fu{
 //子类复写了父类的show方法
 public void show(){
  System.out.println("Zi show");
 }
}
  public  class Test{
 public static void main(String[] args) {
  Zi z = new Zi();
  z.show(); //Zi show 子类有直接使用子类
 }
}
```

08方法覆盖的需求

```
A:方法覆盖的需求
    a:案例:比如手机，当描述一个手机时，它具有发短信，打电话，显示来电号码功能，
    后期由于手机需要在来电显示功能中增加显示姓名和头像，
    这时可以重新定义一个类描述智能手机，并继承原有描述手机的类。
    并在新定义的类中覆盖来电显示功能，在其中增加显示姓名和头像功能

    b:分析:我们不改装(破坏)原来的手机,而是再买一个新的智能手机,不但有原有的功能,而且还有
            例:厂商发布新手机都是基于原有手机的升级,不会拿着原有的手机在卖,新产一款
      1:分析类的构建:
          手机类
            属性(成员变量):无
            行为(成员方法):
                发短信
                打电话
                来电显示:显示来电号码
        智能手机类:
            属性(成员变量):无
            行为(成员方法):
              发短信
              打电话
              来电显示:显示来电号码,显示姓名和头像

        手机类和智能手机类有共性内容:
              发短信
              打电话
              显示来电号码

      2:继承关系分析:
        对于发短信和打电话功能,让智能手机直接沿用(继承)手机的就可以
        但是在智能手机中的来电显示不但实现号码,还显示姓名和头像,同样的都是来电显示功能,
```

09方法覆盖的手机案例实现

```
//手机类
class Phone{
 public void sendMessage(){
  System.out.println("发短信");
 }
 public void call(){
  System.out.println("打电话");
 }
 public void showNum(){
  System.out.println("来电显示号码");
 }
}

//智能手机类
class NewPhone extends Phone{
 //覆盖父类的来电显示号码功能，并增加自己的显示姓名和图片功能
 //从现实生活角度考虑沿用原有的showNum方法名便于用户更快熟悉和接受,而不是再起个新的名字
 //用户还需要花费大量时间慢慢接受

 public void showNum(){
  //调用父类已经存在的功能使用super
            //如果不加super这是调用子类自身的showNum(),自己调用自己,递归
            //方法不断入栈导致内存溢出
  super.showNum();


  //增加自己特有显示姓名和图片功能
  System.out.println("显示来电姓名");
  System.out.println("显示头像");
 }
}

public class Test {
 public static void main(String[] args) {
  new NewPhone().showNum();//来电显示  显示来电姓名 显示头像
 }
}
```

10方法覆盖的注意事项

A:方法覆盖的注意事项
a:权限:子类方法覆盖父类方法，必须要保证权限大于等于父类权限。
四大权限:public>默认=protected>private

```
  class Fu{
 void show(){

 }
 public void method(){
```

```
 }
   }
   class Zi() extends Fu{
 public void show(){//编译运行没问题

 }
 void method(){//编译错误

    }
   }
```
b:方法定义:子类方法和要重写的父类的方法:方法的方法名和参数列表都要一样。
  关于方法的返回值:
    如果是基本数据类型,子类的方法和重写的父类的方法返回值类型必须相同
    如果是引用数据类型,子类的方法和重写的父类的方法返回值类型可以相同或者子类方法的返回
    ```
    class Fu{
 int show(){

 }
 public Fu method(){

 }

 public Fu method2(){

 }

   }
   class Zi() extends Fu{
 public int show(){//返回值为基本类型的重写

 }
 public Fu method(){//子类的方法和重写的父类的方法返回值类型可以相同

    }
    public Zi method2(){//子类方法的返回值类型是父类方法返回值类型的子类

    }
   }
```
c:重载与重写对比:
   重载:
    权限修饰符(public private 默认):无关
    方法名:重载的两个方法的方法名必须相同
    形参列表:
      形参类型的顺序不同
      形参的个数不同
      形参的类型不同
      三者至少满足一个
    返回值类型:
      重载与返回值类型无关
重写:
    权限修饰符(public private 默认):
      子类方法的权限>=父类的方法的权限
```
```

```
    方法名：
      子类方法和父类方法必须相同
    形参列表：
       子类方法和父类方法的形参列表必须相同
    返回值类型：
      基本类数据类型：
        必须相同
      引用数据类型：
       子类方法的返回值类型和父类方法的返回值类型相同
       或者
       子类方法的返回值类型是父类方法的返回值类型的 子类
```

11抽象类的产生

```
  A:抽象类的产生
    a:分析事物时，发现了共性内容，就出现向上抽取。会有这样一种特殊情况，就是方法功能声明
```

###12抽象类的定义格式
A:抽象方法定义的格式：
a:public abstract 返回值类型 方法名(参数);
抽象类定义的格式：
abstract class 类名 {

```
 }
  b:抽象类示例代码：
  /*
 *  定义类开发工程师类
 *    EE开发工程师 ：  工作
 *    Android开发工程师 ： 工作
 *
 *    根据共性进行抽取,然后形成一个父类Develop
 *    定义方法,工作: 怎么工作,具体干什么呀
 *
 *    抽象类,不能实例化对象, 不能new的
 *    不能创建对象的原因:  如果真的让你new了, 对象.调用抽象方法,抽象方法没有主体,根
 *    抽象类使用: 定义类继承抽象类,将抽象方法进行重写,创建子类的对象
 */
public abstract class Develop {
   //定义方法工作方法,但是怎么工作,说不清楚了,讲不明白
 //就不说, 方法没有主体的方法,必须使用关键字abstract修饰
 //抽象的方法,必须存在于抽象的类中,类也必须用abstract修饰
 public abstract void work();
}
```

13抽象类的使用方式

```
A:抽象类的使用方式
 /*
 *  定义类,JavaEE的开发人员
 *  继承抽象类Develop,重写抽象的方法
 */
public class JavaEE extends Develop{
 //重写父类的抽象方法
 //去掉abstract修饰符,加上方法主体
 public void work(){
  System.out.println("JavaEE工程师在开发B/S 软件");

 }
}
/*
 *  定义Android类,继承开发人员类
 *  重写抽象方法
 */
public class Android extends Develop{
     public void work(){
      System.out.println("Android工程师开发手机软件");
     }
}

/*
 *  测试抽象类
 *    创建他的子类的对象,使用子类的对象调用方法
 */
public class Test {
 public static void main(String[] args) {
   JavaEE ee = new JavaEE();
   ee.work();//"JavaEE工程师在开发B/S 软件"

   Android and = new Android();
   and.work();//"Android工程师开发手机软件"
 }
}
```

14抽象类特点

A:抽象类的特点
a:抽象类和抽象方法都需要被abstract修饰。抽象方法一定要定义在抽象类中。
b:抽象类不可以直接创建对象，原因：调用抽象方法没有意义。
c:只有覆盖了抽象类中所有的抽象方法后，其子类才可以创建对象。否则该子类还是一个抽象类。
之所以继承抽象类，更多的是在思想，是面对共性类型操作会更简单。
abstract class A{
public abstract void func();
public abstract void func2();
}

class A2 extends A{//A2把A中的两个抽象方法都重写掉了
//A2类不再是抽象类
public void func(){}
public void func2(){}
}

```
  abstract class A3 extends A{//含有抽象方法的类一定是抽象类
    public void func(){

    }
    //public abstract void func2();//func2相当于被继承下来
  }
```

15抽象类的设计思想 4:40

```
A:抽象类的设计思想
  a:抽象类的作用:继承的体系抽象类,强制子类重写抽象的方法
    抽象员工:
      规定一个方法,work工作
      EE员工,Android员工

       Develop类 抽象类
       abstract work();
           |
    -------------
   |              |
  EE             Android  //是我开发的一员必须工作
  work(){}        work(){}
```

16抽象类的细节

A:抽象类的细节
a:抽象类一定是个父类？
是的，因为不断抽取而来的。
b:抽象类中是否可以不定义抽象方法?
是可以的，那这个抽象类的存在到底有什么意义呢？不让该类创建对象,方法可以直接让子类去使用
(适配器设计模式)
/*
* 抽象类,可以没有抽象方法,可以定义带有方法体的方法
* 让子类继承后,可以直接使用
*/
public abstract class Animal {
public void sleep(){
System.out.println("动物睡觉");
}

```
}
public class Cat extends Animal{

   }

   public class Test {
 public static void main(String[] args) {
  //Cat c = new Cat();
  new Cat().sleep();//不让该类创建对象,方法可以直接让子类去使用
 }
   }
c:抽象关键字abstract不可以和哪些关键字共存？
```

 1:private：私有的方法子类是无法继承到的，也不存在覆盖，
而abstract和private一起使用修饰方法，abstract既要子类去实现这个方法,
而private修饰子类根本无法得到父类这个方法。互相矛盾。

```
   /*
 *   抽象类,可以没有抽象方法,可以定义带有方法体的方法
 *   让子类继承后,可以直接使用
 */
public  abstract class Animal {

     // private abstract void show();
     //抽象方法,需要子类重写, 如果父类方法是私有的,子类继承不了,也就没有了重写
}
```


2:final，暂时不关注，后面学
 3:static，暂时不关注，后面学

17员工案例分析

A:员工案例分析:
a:需求描述:
某IT公司有多名员工，按照员工负责的工作不同，进行了部门的划分（研发部员工、维护部员工）。
研发部根据所需研发的内容不同，又分为JavaEE工程师、Android工程师；
维护部根据所需维护的内容不同，又分为网络维护工程师、硬件维护工程师。

```
公司的每名员工都有他们自己的员工编号、姓名，并要做它们所负责的工作。
 工作内容
 JavaEE工程师：员工号为xxx的 xxx员工，正在研发淘宝网站
 Android工程师：员工号为xxx的 xxx员工，正在研发淘宝手机客户端软件
 网络维护工程师：员工号为xxx的 xxx员工，正在检查网络是否畅通
 硬件维护工程师：员工号为xxx的 xxx员工，正在修复打印机
```

b:继承体系:
员工
|

| |
研发部员工 维护部员工
| |

| | | |
JavaEE工程师 Android工程师 网络维护工程师 硬件维护工程师

c:详细描述:
 根据员工信息的描述，确定每个员工都有员工编号、姓名、要进行工作。
则，把这些共同的属性与功能抽取到父类中（员工类），
关于工作的内容由具体的工程师来进行指定。
 工作内容
 JavaEE工程师：员工号为xxx的 xxx员工，正在研发淘宝网站
 Android工程师：员工号为xxx的 xxx员工，正在研发淘宝手机客户端软件
 网络维护工程师：员工号为xxx的 xxx员工，正在检查网络是否畅通
 硬件维护工程师：员工号为xxx的 xxx员工，正在修复打印机
 创建JavaEE工程师对象，完成工作方法的调用

18员工案例Employee类的编写

A:员工案例Employee类的编写:按照分析的继承体系来逐个实现
```
/*
* 定义员工类
* 内容,都是所有子类的共性抽取
* 属性: 姓名,工号
* 方法: 工作
*/
public abstract class Employee {
private String id;// 员工编号
private String name; // 员工姓名
```

```
  public String getId() {
   return id;
  }
  public void setId(String id) {
   this.id = id;
  }
  public String getName() {
   return name;
  }
  public void setName(String name) {
   this.name = name;
  }

  //工作方法（抽象方法）
  public abstract void work();
 }
```

19员工案例的子类的编写

B:员工案例的子类的编写:

```
/*
* 定义研发员工类
* 属于员工中的一种, 继承员工类
* 抽象类Develop 给自己的员工定义自己有的属性
*/
public abstract class Develop extends Employee{
```

```
}

/*
 *  描述JavaEE开发工程师类
 *    工号,姓名 工作方法
 *  其他的员工,也具备这些共性,抽取到父类中,自己就不需要定义了
 *  是研发部员工的一种,继承研发部类
 */
public class JavaEE extends Develop{
 //重写他父类的父类的抽象方法
 public void work(){
  //调用父类的get方法,获取name,id值
  System.out.println("JavaEE的工程师开发淘宝"+ super.getName()+".."+super.get
 }
}
/*
*定义Android工程师 继承 研发部员工类，重写工作方法
*/
public class Android extends Developer {
  @Override
   public void work() {
 System.out.println("员工号为 " + getId() + " 的 " + getName() + " 员工，正在
  }
}



/*
 *   定义维护员工类,属于员工中的一种
 *   继承员工类
 *   抽象类Maintainer 给自己的员工定义自己有的属性
 */
public abstract class Maintainer extends Employee{

}

/*
 *  描述的是网络维护工程师
 *  属于维护部的员工,继承维护部类
 */
public class Network extends Maintainer{
 public void work(){
  System.out.println("网络工程师在检查网络是否畅通"+super.getName()+"..."+super
 }
}
```

□

```
/*
*定义Hardware硬件维护工程师 继承 维护部员工类，重写工作方法
*/
public class Hardware extends Maintainer {
```

```
@Override
public void work() {
System.out.println("员工号为 " + getId() + " 的 " + getName() + " 员工，正在修复打印机");
}
}
```