章节11-章节20

第11章 课时45：this关键字（调用属性）

```
//在以后的开发中，只要是访问类中的属性前面必须加上"this."

class Book{
  private String title;
  private double price;
  public Book(String t,double p){
    title=t;
    price=p;
  }
  //setter\getter略
  public String getInfo(){
    return"书名："+title+",价格:"+price;
  }
}
public class TestDemo{
  public static void main(String args[]){
  Book bk=new Book("Java开发",89.2);
  System.out.println(bk.getInfo());
  }
}

class Book{
  private String title;
  private double price;
  public Book(String title,double price){
    this.title=title;//程序默认找最近一个{}内的参数，加了this.表示是大类中的属性，而不是此方法中的参数。
    this.price=price;
  }
  //setter\getter略
  public String getInfo(){
    return"书名："+this.title+",价格:"+this.price;//以后的开发中，只要是访问类中的属性，前面必须加this.
  }
}
public class TestDemo{
  public static void main(String args[]){
  Book bk=new Book("Java开发",89.2);
  System.out.println(bk.getInfo());
  }
}
```

第11章 课时46：this关键字（调用方法）

```
class Book{
  private String title;
  private double price;
  public Book(String title,double price){
    this.title=title;//程序默认找最近一个{}内的参数，加了this.表示是大类中的属性，而不是此方法中的参数。
    this.price=price;
  }
  //setter\getter略
  public void print(){
    System.out.println("*******");
  }
  public String getInfo(){
   this. print();//调用本类方法，不加this.也认为是调用本类方法，结果一样。为了代码的严谨性应该加this.
    return"书名："+this.title+",价格:"+this.price;//以后的开发中，只要是访问类中的属性，前面必须加this.
  }
}
public class TestDemo{
  public static void main(String args[]){
  Book bk=new Book("Java开发",89.2);
  System.out.println(bk.getInfo());
  }
}


class Book{
  private String title;
  private double price;
  public Book(){
    System.out.println("一个新的Book类对象产生了");//把这句话想象成一个20行的代码
  }
  public Book(String title){
    System.out.println("一个新的Book类对象产生了");//代码重复，违背写代码原则
    this.title=title;
  }
  public Book(String title,double price){
    System.out.println("一个新的Book类对象产生了");
    this.title=title;
    this.price=price;
  }
  //setter\getter略
  public String getInfo(){
    return"书名："+this.title+",价格:"+this.price;
  }
}
public class TestDemo{
```

```
  public static void main(String args[]){
    Book bk=new Book("Java开发",89.2);
    System.out.println(bk.getInfo());
  }
}


class Book{
  private String title;
  private double price;
  public Book(){
    System.out.println("一个新的Book类对象产生了");
  }
  public Book(String title){
    this();//调用本类中的无参构造
    this.title=title;
  }
  public Book(String title,double price){
    this(title);//调用本类中的单参构造
    this.price=price;
  }
  //setter\getter略
  public String getInfo(){
    return"书名："+this.title+",价格:"+this.price;
  }
}
public class TestDemo{
  public static void main(String args[]){
    Book bk=new Book("Java开发",89.2);
    System.out.println(bk.getInfo());
  }
}//虽然以上实现了构造方法的相互调用，但是存在局限，使用this()调用构造方法形式的代码只能放在构造方法的首行。且构造方法必须留一个出口，也就是至少有一个this()未被调用。


class Emp{
  private int empno;
  private String ename;
  private double sal;
  private String dept;
  public Emp(){
    this.empno=0;
    this.ename="无名氏";
    this.sal=0.0;
    this.dept="未定";
  }
  public Emp(int empno){
    this.empno=empno;
    this.ename="临时工";
    this.sal=800.0;
    this.dept="后勤部";
  }
  public Emp(int empno,String ename){
    this.empno=empno;
    this.ename=ename;
    this.sal=2000.0;
    this.dept="技术部";
  }
  public Emp(int empno,String ename,double sal,String dept){
    this.empno=empno;
    this.ename=ename;
    this.sal=sal;
    this.dept=dept;
  }
  public String getInfo(){
    return"雇员编号："+empno+",雇员姓名:"+ename+",工资:"+sal+",部门:"+dept;
  }
}
public class TestDemo{
  public static void main(String args[]){
    Emp ea=new Emp();
    Emp eb=new Emp(7366);
    Emp ec=new Emp(7566,"ALLEN");
    Emp ed=new Emp(7839,"KING",5000,"财务部");
    System.out.println(ea.getInfo());
    System.out.println(eb.getInfo());
    System.out.println(ec.getInfo());
    System.out.println(ed.getInfo());
  }
}//以上是传统方式实现的，但是存在重复代码。不合格


class Emp{
  private int empno;
  private String ename;
  private double sal;
  private String dept;
  public Emp(){
    this(0,"无名氏",0.0,"未定");
  }
  public Emp(int empno){
```

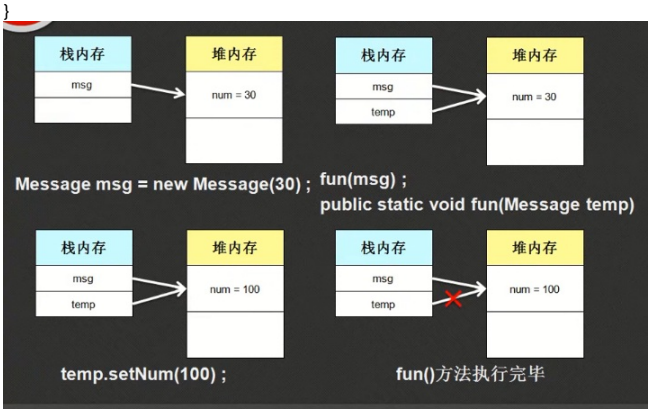
```
    this(empno,"临时工",800.0,"后勤部");
  }
  public Emp(int empno,String ename){
    this(empno,ename,2000.0,"技术部");
  }
  public Emp(int empno,String ename,double sal,String dept){
    this.empno=empno;
    this.ename=ename;
    this.sal=sal;
    this.dept=dept;
  }
  public String getInfo(){
    return"雇员编号："+empno+",雇员姓名:"+ename+",工资:"+sal+",部门:"+dept;
  }
}
public class TestDemo{
  public static void main(String args[]){
    Emp ea=new Emp();
    Emp eb=new Emp(7366);
    Emp ec=new Emp(7566,"ALLEN");
    Emp ed=new Emp(7839,"KING",5000,"财务部");
    System.out.println(ea.getInfo());
    System.out.println(eb.getInfo());
    System.out.println(ec.getInfo());
    System.out.println(ed.getInfo());
  }
}
```

第11章 课时47：this关键字（表示当前对象）

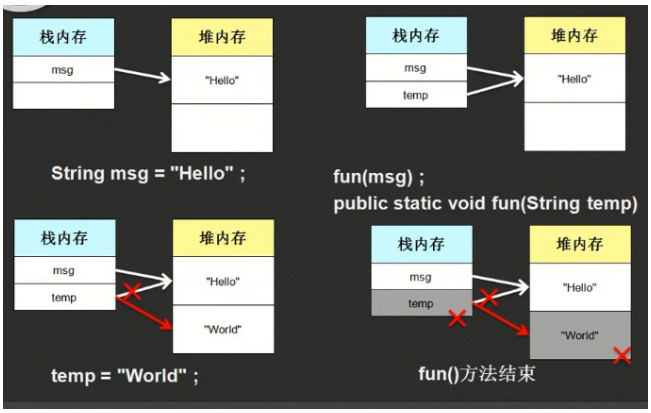
```
class Book{
  public void print(){
    System.out.println("this="+this);//this就是当前调用方法的对象
  }//哪个对象调用了print方法，this就自动与此对象指向同一块内存地址
}
public class TestDemo{
  public static void main(String args[]){
    Book booka=new Book();
    Book bookb=new Book();
    System.out.println("booka"+booka);
    booka.print();
    System.out.println("————————————");
    System.out.println("bookb"+bookb);
    bookb.print();
  }
}
```

第12章 课时48：引用传递

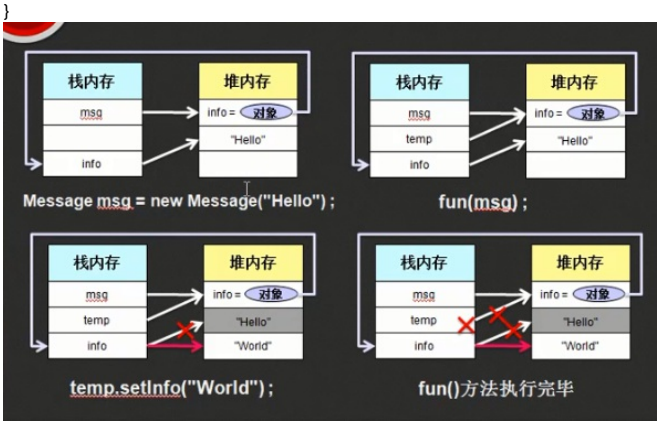
```
//引用传递是Java的精髓所在，引用传递的核心意义是，同一块堆内存空间可以被不同的栈内存所指向。
//范例1：
class Message{
  private int num=10;
  public Message(int num){
    this.num=num;
  }
  public void setNum(int num){
    this.num=num;
  }
  public int getNum(){
    return this.num;
  }
}

public class TestDemo{
  public static void main(String args[]){
    Message msg=new Message(30);
    fun(msg);//引用传递,将msg值传递给temp
    System.out.println(msg.getNum());
  }
  public static void fun(Message temp){
    temp.setNum(100);
  }
}
```



```
//范例2：
public class TestDemo{
  public static void main(String args[]){
    String msg="hello";
    fun(msg);
    System.out.println(msg);
  }
  public static void fun(String temp){
    temp="world";//因为字符串一旦生成就不可更改，所以msg没变化，而temp指向了一块新的堆内存。
  }
}
```




```
//范例3：
class Message{
  private String info="nihao";
  public Message(String info){
    this.info=info;
  }
  public void setInfo(String info){
    this.info=info;
  }
  public String getInfo(){
    return this.info;
  }
}
public class TestDemo{
  public static void main(String args[]){
    Message msg=new Message("Hello");
    fun(msg);
    System.out.println(msg.getInfo());
  }
  public static void fun(Message temp){
  temp.setInfo("World");
  }
}
```



//虽然String属于类，属于引用类型，但是由于其内容不可改变的特点，很多时候就直接把String当成基本数据类型那样使用。

第12章 课时49：引用传递实际应用

```
class Member{
  private int mid;
  private String name;
  private Member child;
//car有实例化对象表示有车
//car为null表示没有车
  private Car car;//表示属于人的车
  public Member(int mid,String name){
    this.mid=mid;
    this.name=name;
  }
  public String getInfo(){
    return "人员编号:"+this.mid+",姓名:"+this.name;
```

```
  }
  public void setChild(Member child){//使用的是自定义类型
    this.child=child;
  }
  public Member getChild(){
    return this.child;
  }
  public void setCar(Car car){
    this.car=car;
  }
  public Car getCar(){
    return this.car;
  }
}
class Car{
  private String pname;
  private Member member;//车属于一个人
  public Car(String pname){
    this.pname=pname;
  }
  public String getInfo(){
    return "车的名字:"+this.pname;
  }
  public void setMember(Member member){
    this.member=member;
  }
  public Member getMember(){
    return this.member;
  }
}
public class TestDemo{
  public static void main(String args[]){
    Member m=new Member(1,"陈冠佑");
    Member chd=new Member(2,"陈冠希");
    Car c=new Car("八手奥拓100");
    Car cc=new Car("法拉利M9");
    m.setCar(c);//一个人有一辆车
    c.setMember(m);//一辆车属于一个人
    chd.setCar(cc);//一个孩子有一辆车
    cc.setMember(chd);//一辆车属于一个孩子
    m.setChild(chd);
    System.out.println(m.getCar().getInfo());//通过人找到车
    System.out.println(c.getMember().getInfo());//通过车找到人
    System.out.println(m.getChild().getInfo());//通过人找到他孩子
    System.out.println(m.getChild().getCar().getInfo()); //通过人找到他孩子车的信息 ，红色标注的叫代码链
  }
}

class Member{
  private int mid;
  private String pname;
  private Car car;
  public Member(int mid,String pname){
    this.mid=mid;
    this.pname=pname;
  }
  public String getInfo(){
    return "人员编号："+this.mid+"，人员姓名："+this.pname;
  }
  public void setCar(Car car){
    this.car=car;
  }
  public Car getCar(){
    return this.car;
  }
}
class Car{
  private String cname;
  private Member member;
  public Car(String cname){
    this.cname=cname;
  }
  public String getInfo(){
    return "汽车名称："+this.cname;
  }
  public void setMember(Member member){
    this.member=member;
  }
  public Member getMember(){
    return this.member;
  }
}
public class TestDemo{
  public static void main(String args[]){
    Member m=new Member(1,"小鱼");
    Car c=new Car("福特野马");
    m.setCar(c);
    c.setMember(m);
    System.out.println(m.getCar().getInfo());
    System.out.println(c.getMember().getInfo());
```

```
  }
}
```

第13章 课时50：综合实战：数据表与简单Java类映射

```
class Dept{
  private int deptno;
  private String dname;
  private String loc;
  private Emp emps [];
  public void setEmps(Emp [] emps){//int data[]=new int[5];举例说明，5是长度。
    this.emps=emps;
  }
  public Emp[] getEmps(){
    return this.emps;
  }
  public Dept(int deptno,String dname,String loc){
    this.deptno=deptno;
    this.dname=dname;
    this.loc=loc;
  }
  public String getInfo(){
    return "部门编号："+this.deptno+"，部门名称："+this.dname+"，位置:"+this.loc;
  }
}
class Emp{
  private int empno;
  private String ename;
  private String job;
  private double sal;
  private double comm;
  private Dept dept;
  private Emp mgr;
  public void setMgr(Emp mgr){
    this.mgr=mgr;
  }
  public Emp getMgr(){
    return this.mgr;
  }
  public void setDept(Dept dept){
    this.dept=dept;
  }
  public Dept getDept(){
    return this.dept;
  }

  public Emp(int empno,String ename,String job,double sal,double comm){
    this.empno=empno;
    this.ename=ename;
    this.job=job;
    this.sal=sal;
    this.comm=comm;
  }
  public String getInfo(){
    return "雇员编号："+this.empno+"，雇员姓名："+this.ename+"，工作"+this.job+"，工资:"+this.sal+"，小费："+this.comm;
  }
}

public class TestDemo{
  public static void main(String args[]){
  //第一步：设置数据
  //1、产生各自的独立对象
    Dept dept=new Dept(10,"ACCOUNTING","New York");//财务部
    Emp ea=new Emp(1234,"SMITH","CLERK",800.0,0.00);
    Emp eb=new Emp(2345,"FORD","MANAGER",2450.0,0.00);
    Emp ec=new Emp(3456,"KING","PRESIDENT",5000.0,0.00);
  //2、设置雇员和领导的关系
    ea.setMgr(eb);
    eb.setMgr(ec);
  //3、设置雇员和部门的关系
    ea.setDept(dept);
    eb.setDept(dept);
    ec.setDept(dept);
    dept.setEmps(new Emp[]{ea,eb,ec});
  //取出数据，根据给定结构取出数据，要求如下：
  //可以根据一个雇员查询他所对应的领导信息和部门信息；
  //可以根据一个部门取出所有雇员以及每个雇员的领导信息；
  //第二部：取出数据
  //1、通过雇员找到领导信息和部门信息
    System.out.println(ea.getInfo());
    System.out.println("\t|- "+ea.getMgr().getInfo());
    System.out.println("\t|- "+ea.getDept().getInfo());
  //2、根据部门找到所有的雇员以及每个雇员的领导信息
    System.out.println("————————————————————————————————————————————————————————————————————————");
    System.out.println(dept.getInfo());
    for(int x=0;x<dept.getEmps().length;x++){
      System.out.println("\t|- "+dept.getEmps()[x].getInfo());
      if(dept.getEmps()[x].getMgr()!=null){
        System.out.println("\t\t|- "+dept.getEmps()[x].getMgr().getInfo());
      }
    }
```

```
  }
}
```

章节13 课时 51 03032_综合实战：一对多映射（省份-城市）

```
class Province{
  private int pid;
  private String name;
  private City cities[];
  public Province(int pid,String name){
  this.pid=pid;
  this.name=name;
  }
  public void setCities(City cities[]){
   this.cities=cities;
  }
  public City[] getCities(){
    return this.cities;
  }
  public String getInfo(){
    return "省份编号："+this.pid+"，名称："+this.name;
  }
}
class City{
  private int cid;
  private String name;
  private Province province;
  public  City(int cid,String name){
    this.cid=cid;
    this.name=name;
  }
  public void setProvince(Province province){
    this.province=province;
  }
  public Province getProvince(){
    return this.province;
  }
  public String getInfo(){
    return "城市编号："+this.cid+"，名称"+this.name;
  }
}
public class TestPc{
  public static void main(String args[]){
    //设置关系数据
    //1、先准备好各自独立的对象；
    Province pro=new Province(1,"河北省");
    City c1=new City(1001,"唐山");
    City c2=new City(1002,"秦皇岛");
    City c3=new City(1003,"石家庄");
    c1.setProvince(pro);
    c2.setProvince(pro);
    c3.setProvince(pro);
    pro.setCities(new City[]{c1,c2,c3});
    //取出关系数据
    System.out.println(c1.getProvince().getInfo());
    for(int x=0;x<pro.getCities().length;x++){
      System.out.println(pro.getCities()[x].getInfo());
    }
  }
}
```

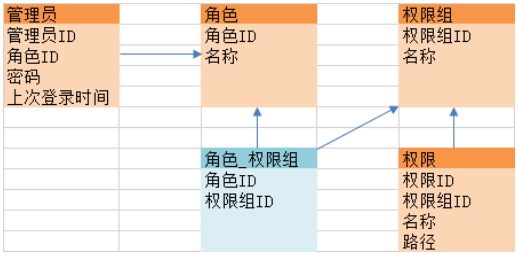
章节13 课时 52 03033_综合实战：双向一对多映射（类型-子类型-商品）

```
class Item{
  private int iid;
  private String name;
  private String note;
  private Subitem subitems[];
  private Product products[];
  public Item(int iid,String name,String note){
    this.iid=iid;
    this.name=name;
    this.note=note;
  }
  public void setSubitems(Subitem subitems[]){
    this.subitems=subitems;
  }
  public Subitem[] getSubitems(){
    return this.subitems;
  }
  public void setProducts(Product products[]){
    this.products=products;
  }
  public Product[] getProducts(){
    return this.products;
  }
  public String getInfo(){
  return "栏目编号："+this.iid+"，名称："+this.name+"，描述："+this.note;
  }
}
class Subitem{
  private int sid;
  private String name;
```

```
  private String note;
  private Item item;
  private Product products[];
  public Subitem(int sid,String name,String note){
    this.sid=sid;
    this.name=name;
    this.note=note;
  }
  public void setItem(Item item){
    this.item=item;
  }
  public Item getItem(){
    return this.item;
  }
  public void setProducts(Product products[]){
    this.products=products;
  }
  public Product[] getProducts(){
    return this.products;
  }
  public String getInfo(){
    return "子栏目编号："+this.sid+"，名称："+this.name+"，描述："+this.note;
  }
}
class Product{
  private int pid;
  private String name;
  private double price;
  private Item item;
  private Subitem subitem;
  public Product(int pid,String name,double price){
    this.pid=pid;
    this.name=name;
    this.price=price;
  }
  public void setItem(Item item){
    this.item=item;
  }
  public Item getItem(){
    return this.item;
  }
  public void setSubitem(Subitem subitem){
    this.subitem=subitem;
  }
  public Subitem getSubitem(){
    return this.subitem;
  }
  public String getInfo(){
    return "产品编号："+this.pid+"，名称："+this.name+"，价格："+this.price;
  }
}
public class TestIsp{
  public static void main(String args[]){
    Item item=new Item(1,"厨房用具","-");
    Subitem suba=new Subitem(1001,"厨具","-");
    Subitem subb=new Subitem(1001,"刀具","-");
    Subitem subc=new Subitem(1001,"餐具","-");
    Product proa=new Product(2001,"菜刀",1000);
    Product prob=new Product(2002,"水果刀",1000);
    Product proc=new Product(3001,"蒸锅",1000);
    Product prod=new Product(3002,"汤锅",1000);
    Product proe=new Product(4001,"青花瓷",1000);
    Product prof=new Product(4002,"竹筷",1000);
    suba.setItem(item);
    subb.setItem(item);
    subc.setItem(item);
    item.setSubitems(new Subitem[]{suba,subb,subc});
    proa.setItem(item);
    prob.setItem(item);
    proc.setItem(item);
    prod.setItem(item);
    proe.setItem(item);
    prof.setItem(item);
    proa.setSubitem(suba);
    prob.setSubitem(suba);
    proc.setSubitem(subb);
    prod.setSubitem(subb);
    proe.setSubitem(subc);
    prof.setSubitem(subc);
    suba.setProducts(new Product[]{proa,prob});
    subb.setProducts(new Product[]{proc,prod});
    subc.setProducts(new Product[]{proe,prof});
    item.setProducts(new Product[]{proa,prob,proc,prod,proe,prof});
  //通过一个类型找到对应的全部子类
    System.out.println(item.getInfo());
    for(int x=0;x<item.getSubitems().length;x++){
      System.out.println("\t|- "+item.getSubitems()[x].getInfo());
    }
    System.out.println("————————————————————————————————————————————————————————————");
  //通过一个类型找到它所对应的全部商品，以及每个商品对应的子类型。
    System.out.println(item.getInfo());
```

```
    for(int y=0;y<item.getProducts().length;y++){
      System.out.println("\t|- "+item.getProducts()[y].getInfo());
      System.out.println("\t\t|- "+item.getProducts()[y].getSubitem().getInfo());
    }
    System.out.println("————————————————————————————————————————————————————————————————————————————————");
//通过一个子类型找到所有对应的全部商品。
    System.out.println(suba.getInfo());
    for(int x=0;x<suba.getProducts().length;x++){
      System.out.println("\t|- "+suba.getProducts()[x].getInfo());
    }
  }
}
```

章节13 课时 53 03034_综合实战：多对多映射（管理员-角色-组-权限）




```
//一个角色有多个管理员，一个管理员属于一个角色
//一个角色有多个权限，一个权限对应多个角色
class Admin{
  private String aid;
  private String password;
  private Role role;
  public Admin(String aid,String password){
    this.aid=aid;
    this.password=password;
  }
  public void setRole(Role role){
    this.role=role;
  }
  public Role getRole(){
    return this.role;
  }
  public String getInfo(){
    return "管理员编号："+this.aid+"，密码："+this.password;
  }
}
class Role{
  private int rid;
  private String title;
  private Admin admins[];
  private Group groups[];
  public Role(int rid,String title){
    this.rid=rid;
    this.title=title;
  }
  public void setAdmins(Admin [] admins){
    this.admins=admins;
  }
  public Admin [] getAdmins(){
    return this.admins;
  }
  public void setGroups(Group [] groups){
    this.groups=groups;
  }
  public Group[] getGroups(){
    return this.groups;
  }

  public String getInfo(){
    return "角色编号："+this.rid+"，角色名称："+this.title;
  }
}
class Group{
  private int gid;
  private String title;
  private Role roles[];
  private Action actions[];
  public Group(int gid,String title){
    this.gid=gid;
    this.title=title;
  }
  public void setRoles(Role [] roles){
    this.roles=roles;
  }
  public Role[] getRoles(){
    return this.roles;
  }
  public void setActions(Action [] actions){
   this.actions=actions;
```

```
  }
  public Action [] getActions(){
    return this.actions;
  }
  public String getInfo(){
    return "权限组编号：" +this.gid+"，名称：" +this.title;
  }
}
class Action{
  private int aid;
  private String title;
  private String url;
  private Group group;
  public Action(int aid,String title,String url){
    this.aid=aid;
    this.title=title;
    this.url=url;
  }
  public void setGroup(Group group){
    this.group=group;
  }
  public Group getGroup(){
   return this.group;
  }
  public String getInfo(){
    return "权限编号：" +this.aid+"，名称：" +this.title+"，路径：" +this.url;
  }
}
public class TestAdmin{
  public static void main(String args[]){
    Admin a1=new Admin("admin","hello");
    Admin a2=new Admin("mldn","hello");
    Admin a3=new Admin("ayou","hello");
    Role r1=new Role(1,"系统管理员");
    Role r2=new Role(2,"信息管理员");
    Group g1=new Group(10,"信息管理");
    Group g2=new Group(11,"用户管理");
    Group g3=new Group(12,"数据管理");
    Group g4=new Group(13,"接口管理");
    Group g5=new Group(14,"备份管理");
    Action ac01=new Action(1001,"新闻发布","-");
    Action ac02=new Action(1002,"新闻列表","-");
    Action ac03=new Action(1003,"新闻审核","-");
    Action ac04=new Action(1004,"增加用户","-");
    Action ac05=new Action(1005,"用户列表","-");
    Action ac06=new Action(1006,"登录日志","-");
    Action ac07=new Action(1007,"雇员数据","-");
    Action ac08=new Action(1008,"部门数据","-");
    Action ac09=new Action(1009,"公司数据","-");
    Action ac10=new Action(1010,"服务传输","-");
    Action ac11=new Action(1011,"短信平台","-");
    Action ac12=new Action(1012,"全部备份","-");
    Action ac13=new Action(1013,"局部备份","-");
    //设置对象的基本关系
    //设置管理员与角色
    a1.setRole(r1);
    a2.setRole(r2);
    a3.setRole(r2);
    r1.setAdmins(new Admin[]{a1});
    r2.setAdmins(new Admin[]{a2,a3});
    //设置角色与管理组
    r1.setGroups(new Group[]{g1,g2,g3,g4,g5});
    r2.setGroups(new Group[]{g1,g2});
    g1.setRoles(new Role[]{r1,r2});
    g2.setRoles(new Role[]{r1,r2});
    g3.setRoles(new Role[]{r1});
    g4.setRoles(new Role[]{r1});
    g5.setRoles(new Role[]{r1});
    //管理员组与权限
    g1.setActions(new Action[]{ac01,ac02,ac03});
    g2.setActions(new Action[]{ac04,ac06,ac06});
    g3.setActions(new Action[]{ac07,ac08,ac09});
    g4.setActions(new Action[]{ac10,ac11});
    g5.setActions(new Action[]{ac12,ac13});
    ac01.setGroup(g1);
    ac02.setGroup(g1);
    ac03.setGroup(g1);
    ac04.setGroup(g2);
    ac05.setGroup(g2);
    ac06.setGroup(g2);
    ac07.setGroup(g3);
    ac08.setGroup(g3);
    ac09.setGroup(g3);
    ac10.setGroup(g4);
    ac11.setGroup(g4);
    ac12.setGroup(g5);
    ac13.setGroup(g5);
    //取出数据
    System.out.println("————————————————————————————————————————————————————————————");
    //可以根据一个管理员找到他所对应的角色，以及每个角色包含的所有权限组的信息，每个权限组所包含所有权限的内容。
    System.out.println(a1.getInfo());
```

```
    System.out.println("\t|- "+a1.getRole().getInfo());
    for(int x=0;x<a1.getRole().getGroups().length;x++){
      System.out.println("\t\t|- "+a1.getRole().getGroups()[x].getInfo());
      for(int y=0;y<a1.getRole().getGroups()[x].getActions().length;y++){System.out.println("\t\t\t|- "+a1.getRole().getGroups()[x].getActions()[y].getInfo());
      }
    }
    System.out.println("————————————————————————————————————————————————————————");
    //根据一个权限组找到具备此权限组的角色以及每个角色所拥有的管理员信息
    System.out.println(g2.getInfo());
    for(int x=0;x<g2.getRoles().length;x++){
      System.out.println("\t|- "+g2.getRoles()[x].getInfo());
      for(int y=0;y<g2.getRoles()[x].getAdmins().length;y++){
        System.out.println("\t\t|- "+g2.getRoles()[x].getAdmins()[y].getInfo());
      }
    }
  }
}
```

章节14 课时 54 03035_对象比较

```
class Book{
  private String title;
  private double price;
  public Book(String title,double price){
    this.title=title;
    this.price=price;
  }
  public boolean compare(Book book){//本类接收本类对象，对象可以直接访问属性。1带回了需要信息，2方便访问。
    if(this==book)//内存地址相同
      return true;
    }
    if(this.title.equals(book.title)&&this.price==book.price){//但钱对象this（调用方法对象，就是b1引用），传递的对象book(引用传递，就是b2引用)
      return true;
    }else{
      return false;
    }
  }
  public String getTitle(){
    return this.title;
  }
  public double getPrice(){
    return this.price;
  }
}
public class TestDemo{
  public static void main(String args[]){
    Book b1=new Book("Java开发",79.8);
    Book b2=new Book("Java开发",79.8);
    if(b1.compare(b2)){
      System.out.println("是同一个对象");
    }else{
      System.out.println("不是同一个对象");
    }
  }
}
```

章节15 课时 55 03036_static关键字（定义属性）

```
//1、static定义属性
class Book{
  private String title;
  private double price;
  static String pub="清华大学出版社";
  public Book(String title,double price){
    this.title=title;
    this.price=price;
  }
  public String getInfo(){
    return "图书名称："+this.title+"，价格："+this.price+"，出版社："+this.pub;
  }
}
public class TestDemo{
  public static void main(String args[]){
    Book ba=new Book("Java开发",10.2);
    Book bb=new Book("Android开发",10.2);
    Book bc=new Book("Oracle开发",10.2);
    Book.pub="北京大学出版社";
    System.out.println(ba.getInfo());
    System.out.println(bb.getInfo());
    System.out.println(bc.getInfo());
  }
}



//所有的非static属性必须产生实例化才可以使用，但static不需要。可以直接被类调用
class Book{
  private String title;
  private double price;
```

```
  static String pub="清华大学出版社";
  public Book(String title,double price){
    this.title=title;
    this.price=price;
  }
  public String getInfo(){
    return "图书名称：" +this.title+ "，价格：" +this.price+ "，出版社：" +this.pub;
  }
}
public class TestDemo{
  public static void main(String args[]){
    System.out.println(Book.pub);
    Book.pub="北京大学出版社";
    System.out.println(Book.pub);
    Book ba=new Book("Java开发",10.9);
    System.out.println(ba.getInfo());
  }
}
```

章节15 课时 56 03037_static关键字（定义方法）

```
//2、static 定义方法
//static定义的属性和方法都不受到实例化对象的控制，也就是说都属于独立类的功能。但是这个时候就会出现一个特别麻烦的问题：此时类中的方法就变为了2组：static和非static方法。这两组方法间的访问也将受到限制
//static方法不能直接访问非static属性或者方法。只能调用static属性或方法。

class Book{
  private boolean flag;
  public Book(boolean flag){
    this.flag=flag;
  }
  public void fun(){
    if(this.flag){
      System.out.println("可以操作！");
    }else{
      System.out.println("不能够操作！");
    }
  }
}
public class TestDemo{
  public static void main(String args[]){
    Book ba=new Book(true);
    Book bb=new Book(false);
    ba.fun();
    bb.fun();
  }
}


//此时MyMath类没有属性，产生对象完全没有意义，所以会使用static方法。写一个类的时候有限考虑的一定是非static方法或属性。
class MyMath{
  public static int add(int x,int y){
    return x+y;
  }
}
public class TestDemo{
  public static void main(String args[]){
    System.out.println(MyMath.add(10,20));
  }
}
```

章节15 课时 57 03038_static关键字（主方法）

```
//public:主方法是程序的开始，所以这个方法对任何操作都一定是可见的。
//static：证明此方法是由类名称调用的。
//void：主方法是一切执行的开始点，既然是所有的开头，就不能回头。
//main ：是一个系统规定好的方法名称，不能修改
//String args[]：指的是程序运行的时候传递的参数。
//范例：
public class TestDemo{
  public static void main(String args[]){
    for(int x=0;x<args.length;x++){
      System.out.println(args[x]);
    }
  }
}
//执行时所有输入的参数必须使用空格分割，例如java TestDemo 参数 参数 参数
//也可以使用双引号来区分  java TestDemo "Hello world" "Hello mldn"
```

章节15 课时 58 03039_static关键字（应用案例）

```
//1、开发之中首选的属性一定不是static属性，首选的方法一定不是static方法；
//2、static属性和方法可以在没有实例化对象的时候直接由类名称进行调用；
//3、static属性保存在全局数据区，而内存区共有4个：栈内存、堆内存、全局数据区、全局代码区。
class Book{
  private String title;
  private static int num=0;
  public Book(){
    this("notitle-"+num++);
  }
  public Book(String title){
    this.title=title;
```

```
  }
  public String getTitle(){
    return this.title;
  }
}
public class TestDemo{
  public static void main(String args[]){
    System.out.println(new Book("Java").getTitle());
    System.out.println(new Book().getTitle());
    System.out.println(new Book().getTitle());
  }
}
```

章节16 课时 59 03040_代码块（普通代码块）

```
public class TestDemo{
  public static void main(String args[]){
    //if(true){//条件一定满足，去除此行的判断，大括号里面的内容就变为普通代码块
      int num=10;//局部变量
      System.out.println("num="+num);
    }
    int num=100;//全局变量
    System.out.println("num="+num);
  }
}
```

章节16 课时 60 03041_代码块（构造块）

```
//如果一个代码块写在一个类里面，就称为构造代码块
class Book{
  public Book(){
  System.out.println("[A]Book类的构造方法");
  }
  {
    System.out.println("[B]Book类中的构造块");
  }
}
public class TestDemo{
  public static void main(String args[]){
    new Book();
    new Book();
    new Book();
  }
}
```

章节16 课时 61 03042_代码块（静态块）

```
//如果一个代码块使用了static进行定义的话，那么就称为静态块。可是静态块的使用要分为2种情况：
//非主类中使用,优先于构造块执行，不管有多少个实例化对象，静态块只执行一次。它的功能是为了类中的static属性初始化的。
class Book{
  static String msg;
  public Book(){
  System.out.println("[A]Book类的构造方法");
  }
  {
    System.out.println("[B]Book类中的构造块");
  }
  static{
    msg="Hello".substring(0,2);
    System.out.println("[C]Book类中的静态块");
  }
}
public class TestDemo{
  public static void main(String args[]){
    new Book();
    new Book();
    new Book();
    System.out.println(Book.msg);
  }
}

//情况2：在主类中使用，静态块优先于主方法执行

public class TestDemo{
  static{
    System.out.println("**************");
  }
  public static void main(String args[]){
    System.out.println("Hello World!");
  }
}
//总结，代码块能不用就别用，唯一还可用的就是静态块
```

章节17 课时 62 03043_内部类（基本概念）

```
class Outer{
  private String msg="Hello World!";
  class Inner{//定义了一个内部类
    public void print(){
      System.out.println(msg);
    }
  }
  public void fun(){
```

```
    new Inner().print();//实例化内部类对象，并且调用print()方法。
  }
}
public class TestDemo{
  public static void main(String args[]){
    Outer out=new Outer();//实例化外部类对象
    out.fun();//调用外部类方法
  }
}

//如果把内部类Inner拿出来，但是要实现同样的功能。
//下列代码折腾了半天，最终目的只有一个，让内部类可以访问外部类中定义的一个私有的msg属性内容。

class Outer{
  private String msg="Hello World!";
  public String getMsg(){
    return this.msg;
  }
  public void fun(){
    new Inner(this).print();//实例化内部类对象，并且调用print()方法。
  }
}
class Inner{
  private Outer out;
  public Inner(Outer out){
   this.out=out;
  }
  public void print(){
    System.out.println(this.out.getMsg());
  }
}
public class TestDemo{
  public static void main(String args[]){

    Outer out=new Outer();//实例化外部类对象
    out.fun();//调用外部类方法
  }
}
//内部类有一个最大的优点，可以方便的访问外部类的私有操作。但是，反之外部类也可以访问内部类的私有属性。
//范例：

class Outer{
  private String msg="Hello World!";
  class Inner{//定义了一个内部类
    private String info="世界，你好！";
    public void print(){
      System.out.println(msg);
    }
  }
  public void fun(){
    Inner in=new Inner();//内部类对象
    System.out.println(in.info);//直接利用了内部类对象访问了内部类的私有属性
  }
}
public class TestDemo{
  public static void main(String args[]){
    Outer out=new Outer();//实例化外部类对象
    out.fun();//调用外部类方法
  }
}


class Outer{
  private String msg="Hello World!";
  class Inner{//定义了一个内部类
    public void print(){
      System.out.println(Outer.this.msg);//外部类的当前对象，不加Outer只用this.就只是本类。
    }
  }
  public void fun(){
  new Inner().print();//内部类对象
    System.out.println(in.info);//直接利用了内部类对象访问了内部类的私有属性
  }
}
public class TestDemo{
  public static void main(String args[]){
    Outer out=new Outer();//实例化外部类对象
    out.fun();//调用外部类方法
  }
}

//外部类.内部类  对象=new  外部类().new  内部类()
//上述表达式为：内部类对象（在外部）的实例化语法
//范例：内部类对象实例化,一定先实例化外部类对象

class Outer{
  private String msg="Hello World!";
  class Inner{//定义了一个内部类
    public void print(){
      System.out.println(Outer.this.msg);//
    }
```

```
  }
}
public class TestDemo{
  public static void main(String args[]){
    Outer.Inner in=new Outer().new Inner();//实例化内部类对象
    in.print();
  }
}

//如果一个内部类只希望被一个外部类访问，不希望被外部调用，可以在前面加private，例如：
class Outer{
  private String msg="Hello World!";
  private class Inner{//定义了一个内部类
    public void print(){
      System.out.println(Outer.this.msg);//
    }
  }
}
public class TestDemo{
  public static void main(String args[]){
    //Outer.Inner in=new Outer().new Inner();//此句编译时会出错，因为无法调用。
    //in.print();
  }
}
```

章节17 课时 63 03044_内部类（static定义内部类）

```
//使用static定义的属性或者方法是不受到实例化对象控制的。如果使用static定义内部类，那么它不会受到外部类的实例化对象控制。
//如果一个内部类使用了static定义的话，那么这个内部类就变为了外部类，且只能访问外部类中定义的static操作。相当于定义了一个外部类。
class Outer{
  private static String msg="Hello World!";//如果不加static，则内部类无法调用此属性。因为内部类前加了static。
  static class Inner{//使用static定义了一个内部类
    public void print(){
      System.out.println(msg);
    }
  }
}
public class TestDemo{
  public static void main(String args[]){
    Outer.Inner in=new Outer.Inner();//此时不再需要先产生外部类对象，再产生内部类对象，仿佛成了一个独立的类。
    in.print();
  }
}
```

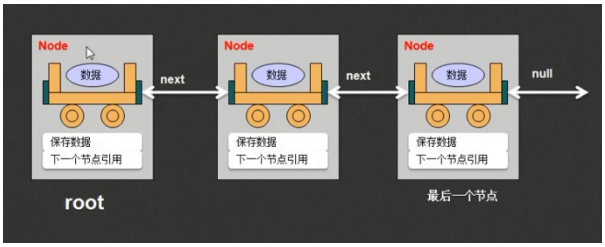
章节17 课时 64 03045_内部类（方法中定义内部类）

```
//内部类只是阐述了基本定义形式，但是没有讲解如何去使用的
//2、内部类可以与外部类之间方便的进行私有属性的访问
//3、内部类可以使用private声明，声明之后无法再外部实例化背部类对象。
//语法：外部类.内部类  内部类对象=new 外部类().new内部类();
//4、使用static定义的内部类就相当于一个外部类。
//语法：外部类.内部类  内部类对象=new 外部类.内部类();
//5、内部类可以在方法中定义。

class Outer{
  private String msg="Hello World!";
  public void fun(){
    class Inner{//方法中定义内部类
      public void print(){
        System.out.println("属性："+Outer.this.msg);
      }
    }
    new Inner().print();//实例化Inner类对象，并调用print方法。
  }
}
public class TestDemo{
  public static void main(String args[]){
    new Outer().fun();//实例化Outer类对象，并调用fun()方法。
  }
}
//方法中定义内部类如上，但是方法里面会接受参数，也会定义变量。上述代码可以访问吗？请看下方：
class Outer{
  private String msg="Hello World!";
  public void fun(int num){//方法参数
    double score=99.9;//方法变量
    class Inner{//方法中定义内部类
      public void print(){
        System.out.println("属性："+Outer.this.msg);
        System.out.println("方法参数："+num);
        System.out.println("方法变量："+score);
      }
    }
    new Inner().print();//实例化Inner类对象，并调用print方法。
  }
}
public class TestDemo{
  public static void main(String args[]){
    new Outer().fun(100);//实例化Outer类对象，并调用fun()方法。
  }
}
```

//此时发现没有加入任何的修饰，方法中的内部类可以访问方法的参数以及定义的变量，但是这种操作只适合于JDK18之后的版本。1.7及之前的版本有一个严格要求：方法中定义的内部类如果想要访问方法的参数或者方法定义的变量，那么参数或变量前一定要加上"final"标记。

```
class Outer{
  private String msg="Hello World!";
  public void fun(final int num){//方法参数
    final double score=99.9;//方法变量
    class Inner{//方法中定义内部类
      public void print(){
        System.out.println("属性："+Outer.this.msg);
        System.out.println("方法参数："+num);
        System.out.println("方法变量："+score);
      }
    }
    new Inner().print();//实例化Inner类对象，并调用print方法。
  }
}
public class TestDemo{
  public static void main(String args[]){
    new Outer().fun(100);//实例化Outer类对象，并调用fun()方法。
  }
}
```

章节18 课时 65 03046_链表的定义与使用（链表基本概念）




```
class Node{
  private String data;
  private Node next;
  public Node(String data){
    this.data=data;
  }
  public void setNext(Node next){
    this.next=next;
  }
  public Node getNext(){
    return this.next;
  }
  public String getData(){
    return this.data;
  }
}
public class LinkDemo{
  public static void main(String args[]){
    Node root=new Node("火车头");
    Node n1=new Node("车厢A");
    Node n2=new Node("车厢B");
    root.setNext(n1);
    n1.setNext(n2);
    //取出所有数据
    Node currentNode=root;//当前从根节点开始读取
    while(currentNode!=null){//当前节点存在有数据
      System.out.println(currentNode.getData());
      currentNode=currentNode.getNext();//将下一个节点设置为当前节点
    }
  }
}
```

```
//使用递归方法
class Node{
  private String data;
  private Node next;
  public Node(String data){
    this.data=data;
  }
  public void setNext(Node next){
    this.next=next;
  }
  public Node getNext(){
    return this.next;
  }
  public String getData(){
    return this.data;
  }
}
public class LinkDemo{
  public static void main(String args[]){
    Node root=new Node("火车头");
    Node n1=new Node("车厢A");
```

```
    Node n2=new Node("车厢B");
    root.setNext(n1);
    n1.setNext(n2);
    print(root);
  }
  public static void print(Node current){
    if(current==null){//递归结束条件
      return;//结束方法
    }
    System.out.println(current.getData());
    print(current.getNext());
  }
}//两种方法的区别，取数据的方式不一样，第一种是直接定义一个对象作为临时数据存储，也就是当前数据。第二种是定义一个类属性，通过print（）；方法调用。
```

章节18 课时 66 03047_链表的定义与使用（链表基本实现）

```
class Node{
  private String data;
  private Node next;
  public Node(String data){
    this.data=data;
  }
  public void setNext(Node next){
    this.next=next;
  }
  public Node getNext(){
    return this.next;
  }
  public String getData(){
    return this.data;
  }
//实现节点的添加
//第一次调用(Link):this=Link.root
//第二次调用(Node):this=Link.root.next
//第三次调用(Node):this=Link.root.next.next
  public void addNode(Node newNode){
    if(this.next==null){//当前节点的下一个为null;
      this.next=newNode;//保存新节点
    }else{
      this.next.addNode(newNode);
    }
  }
//第一次调用(Link):this=Link.root
//第二次调用(Link):this=Link.root.next
//第三次调用(Link):this=Link.root.next.next
  public void printNode(){
    System.out.println(this.data);//输出当前节点数据
    if(this.next!=null){//现在还有下一个节点
      this.next.printNode();//输出下一个
    }
  }
}
class Link{//负责数据的设置和输出
  private Node root;//根节点
  public void add(String data){//增加数据
    Node newNode=new Node(data); //为了可以设置数据的先后关系，所以将data包装在一个Node类对象里。
    if(this.root==null){//一个链表只有一个根节点
      this.root=newNode;//将新的节点设置为根节点
    }else{//根节点已经存在了
      this.root.addNode(newNode);//随后后面增加的元素应交由节点来决定。
    }
  }
  public void print(){//输出数据
    if(this.root!=null){//现在存在根节点
      this.root.printNode();//交给Node类输出
    }
  }
}
public class LinkDemo{
  public static void main(String args[]){
    Link link=new Link();//由这个类负责所有的数据操作
    link.add("Hello"); //存放数据
    link.add("World"); //存放数据
    link.add("mldn");  //存放数据
    link.add("www");  //存放数据
    link.print(); //展示数据
  }
}
//Link类的主要功能是控制Node类对象产生和根节点
//Node类主要负责数据的保存以及引用关系的分配
//上述代码执行步骤————————————————————————————————————————
```

//Node 负责节点，看到Node的属性先想到节点。Link 负责增加和输出。

//执行顺序：link.add("Hello");

//→public void add(String data)开始增加数据

//→this.root=newNode; 或者this.root.addNode(newNode); 从根节点开始，或 如果此节点已有数据，往下增加新节点。

//→public void addNode(Node newNode)增加新节点的方法，增加节点的对象。

//→第一次调用 this=Link.root

```
if(this.next==null){//当前节点的下一个为null;
      this.next=newNode;//保存新节点
}else{//当前节点之后还存在有节点，将值赋给新节点，如果节点后的节点之后还有节点怎么办？用递归
//第二次调用(Node):this=Link.root.next
//第三次调用(Node):this=Link.root.next.next
      this.next.addNode(newNode);
//→保存完毕该输出了public void print(){//输出数据
    if(this.root!=null){//现在存在根节点
      this.root.printNode();//交给Node类输出
}
//→public void printNode(){
//第一次调用(Link):this=Link.root
System.out.println(this.data);//输出当前节点数据
//第二次调用(Link):this=Link.root.next
//第三次调用(Link):this=Link.root.next.next
    if(this.next!=null){//现在还有下一个节点
      this.next.printNode();//用递归
```

输出下一个，下下个直到结束。

章节18 课时 67 03048_链表的定义与使用（确定程序结构）

//在开发具体的可用链表操作之前，首先必须明确一个道理：Node类负责所有节点数据的保存以及节点关系的匹配，所以Node类不可能单独去使用，而以上的实现里Node类是可以单独使用的，外部可以绕过Link类直接操作Node类，这样明显是没有任何意义存在的。所以下面必须修改设计结构，让Node类只能被Link类使用。

//这个时候使用内部类明显是一个最好的选择。内部类可以使用private定义，这样一个内部类只能被一个外部类使用，另外一点，内部类可以方便与外部类之间进行私有属性的直接访问。

//范例：链表的开发结构

```
class Link{//链表类，外部能够看见的只有这一个类
  private class Node{//定义节点类，之所以定义在内部，主要是为Link类服务。
    private String data;
    private Node next;
    public Node(String data){
      this.data=data;
    }
  }
  //以上为内部类
    private Node root;//需要根节点
}
public class LinkDemo{
  public static void main(String args[]){
    Node n=new Node("a");
  }
}
//随后主要就是进行代码的填充以及功能的完善。
```

章节18 课时 68 03049_链表的定义与使用（增加数据）

```
//如果要进行新数据的增加，则应该有Link类负责节点对象的产生，并且由Link类维护根节点，所有的节点的关系匹配交给Node类进行处理。
class Link{//链表类，外部能够看见的只有这一个类
  private class Node{//定义节点类，之所以定义在内部，主要是为Link类服务。
    private String data;
    private Node next;
    public Node(String data){
      this.data=data;
    }
    public void addNode(Node newNode){
      if(this.next==null){
        this.next=newNode;
      }else{//向后继续保存
        this.next.addNode(newNode);
      }
    }
  }
  //以上为内部类
  private Node root;//需要根节点
  public void add(String data){//假设不允许有空值
    if(data==null){
      return;
    }
    Node newNode=new Node(data);
    if(this.root==null){//当前没有根节点
      this.root=newNode;//保存根节点
    }else{//根节点存在，其他节点交给Node类处理
      this.root.addNode(newNode);
```

```
    }
  }
}
public class LinkDemo{
  public static void main(String args[]){
    Link all=new Link();
    all.add("Hello");
    all.add("World");
    all.add(null);
  }
}
//随后主要就是进行代码的填充以及功能的完善。
//此时使用了一个不许为null的判断，但并不是所有的链表都不许为null。
```

章节18 课时 69 03050_链表的定义与使用（取得链表长度）

```
//既然每一个链表对象都只有一个root根元素，那么每一个链表就有自己的长度，可以直接在Link 类里面设置一个count属性，随后每一次数据添加完成之后，可以进行个数的自增。
//范例：1、增加一个count属性。2、在add（）方法里面增加数据的统计操作。3、随后为Link类增加一个size()方法
class Link{//链表类，外部能够看见的只有这一个类
  private class Node{//定义节点类，之所以定义在内部，主要是为Link类服务。
    private String data;
    private Node next;
    public Node(String data){
      this.data=data;
    }
    public void addNode(Node newNode){
      if(this.next==null){
        this.next=newNode;
      }else{//向后继续保存
        this.next.addNode(newNode);
      }
    }
  }
  //以上为内部类
  private Node root;//需要根节点
  private int count=0;//保存元素的个数
  public void add(String data){//假设不允许有空值
    if(data==null){
      return;
    }
    Node newNode=new Node(data);
    if(this.root==null){//当前没有根节点
      this.root=newNode;//保存根节点
    }else{//根节点存在，其他节点交给Node类处理
      this.root.addNode(newNode);
    }
    this.count++;//每次保存完成后，数据加1
  }
  public int size(){
    return this.count;
  }
}
public class LinkDemo{
  public static void main(String args[]){
    Link all=new Link();
    all.add("Hello");
    all.add("World");
    all.add(null);
    System.out.println(all.size());
  }
}
//随后主要就是进行代码的填充以及功能的完善。
//此时使用了一个不许为null的判断，但并不是所有的链表都不许为null。
```

章节18 课时 70 03051_链表的定义与使用（判断空链表）

```
//判断是否为空链表可以通过2种方式完成：1、判断root有对象(是否为空)，2、判断保存的数据量
class Link{//链表类，外部能够看见的只有这一个类
  private class Node{//定义节点类，之所以定义在内部，主要是为Link类服务。
    private String data;
    private Node next;
    public Node(String data){
      this.data=data;
    }
    public void addNode(Node newNode){
      if(this.next==null){
        this.next=newNode;
      }else{//向后继续保存
        this.next.addNode(newNode);
      }
    }
  }
  //以上为内部类
  private Node root;//需要根节点
  private int count=0;//保存元素的个数
  public void add(String data){//假设不允许有空值
    if(data==null){
      return;
```

```
    }
    Node newNode=new Node(data);
    if(this.root==null){//当前没有根节点
      this.root=newNode;//保存根节点
    }else{//根节点存在，其他节点交给Node类处理
      this.root.addNode(newNode);
    }
    this.count++;//每次保存完成后，数据加1
  }
  public int size(){
    return this.count;
  }
  public boolean isEmpty(){
    return this.count==0;
  }
}
public class LinkDemo{
  public static void main(String args[]){
    Link all=new Link();
    System.out.println(all.isEmpty());
    all.add("Hello");
    all.add("World");
    all.add(null);
    System.out.println(all.size());
    System.out.println(all.isEmpty());
  }
}
//随后主要就是进行代码的填充以及功能的完善。
//此时使用了一个不许为null的判断，但并不是所有的链表都不许为null。
```

章节18 课时 71 03052_链表的定义与使用（内容查询）

```
class Link{//链表类，外部能够看见的只有这一个类
  private class Node{//定义节点类，之所以定义在内部，主要是为Link类服务。
    private String data;
    private Node next;
    public Node(String data){
      this.data=data;
    }
    public void addNode(Node newNode){
      if(this.next==null){
        this.next=newNode;
      }else{//向后继续保存
        this.next.addNode(newNode);
      }
    }
    //第一次调用(Link):this=Link.root
    //第二次调用(Node):this=Link.root.next
    public boolean containsNode(String data){
      if(data.equals(this.data)){//当前节点数据为要查询的数据
        return true;//后面不再查询了
      }else{//当前节点数据不满足查询
        if(this.next!=null){//有后续节点
          return this.next.containsNode(data);
        }else{//没有后续节点
          return false;//没得查了
        }
      }
    }
  }
  //以上为内部类
  private Node root;//需要根节点
  private int count=0;//保存元素的个数
  public void add(String data){//假设不允许有空值
    if(data==null){
      return;
    }
    Node newNode=new Node(data);
    if(this.root==null){//当前没有根节点
      this.root=newNode;//保存根节点
    }else{//根节点存在，其他节点交给Node类处理
      this.root.addNode(newNode);
    }
    this.count++;//每次保存完成后，数据加1
  }
  public int size(){
    return this.count;
  }
  public boolean isEmpty(){
    return this.count==0;
  }
  public boolean contains(String data){
    if(data==null||this.root==null){//现在没有要查询的数据，根节点也不保存数据
      return false;
    }
    return this.root.containsNode(data);
  }
}
public class LinkDemo{
```

```
  public static void main(String args[]){
    Link all=new Link();
    all.add("Hello");
    all.add("World");
    System.out.println(all.contains("Hello"));
    System.out.println(all.contains("ayou"));
  }
}
//随后主要就是进行代码的填充以及功能的完善。
//此时使用了一个不许为null的判断，但并不是所有的链表都不许为null。
```

章节18 课时 72 03053_链表的定义与使用（根据索引取得数据）

```
class Link{
  private class Node{
    private String data;
    private Node next;
    public Node(String data){
      this.data=data;
    }
    public void addNode(Node newNode){
      if(this.next==null){
        this.next=newNode;
      }else{
        this.next.addNode(newNode);
      }
    }
    public boolean containsNode(String data){
      if(data.equals(this.data)){
        return true;
      }else{
        if(this.next!=null){
          return this.next.containsNode(data);
        }else{
          return false;
        }
      }
    }
    public String getNode(int index){
      if(Link.this.foot++==index){//使用当前的foot内容与要查询的索引进行比较，随后将foot的内容自增是为了下次查询方便
        return this.data;
      }else{
        return this.next.getNode(index);
      }
    }
  }
  private Node root;
  private int count=0;
  private int foot=0;
  public void add(String data){
    if(data==null){
      return;
    }
    Node newNode=new Node(data);
    if(this.root==null){
      this.root=newNode;
    }else{
      this.root.addNode(newNode);
    }
    count++;
  }
  public int size(){
    return this.count;
  }
  public boolean isEmpty(){
    return this.count==0;
  }
  public String get(int index){
    if(index>=this.count){//超过了查询范围
      return null;//没有数据
    }
    this.foot=0;//表示从前向后查询
    return this.root.getNode(index);//将查询过程交给Node类处理
  }
  public boolean contains(String data){
    if(data==null||this.root==null){
      return false;
    }else{
      return this.root.containsNode(data);
    }
  }
}
public class LinkDemo{
  public static void main(String args[]){
    Link all=new Link();
    System.out.println(all.size());
    System.out.println(all.isEmpty());
    all.add("Hello");
    all.add("World");
```

```
    all.add(null);
    System.out.println(all.isEmpty());
    System.out.println(all.contains("Hello"));
    System.out.println(all.contains("ayou"));
    System.out.println(all.get(0));
    System.out.println(all.get(1));
    System.out.println(all.get(2));//根据索引查询结果为null
    System.out.println(all.get(3));//根据索引查询结果为null
  }
}
```

章节18 课时 73 03054_链表的定义与使用（修改链表数据）

```
class Link{
  private class Node{
    private String data;
    private Node next;
    public Node(String data){
      this.data=data;
    }
    public void addNode(Node newNode){
      if(this.next==null){
        this.next=newNode;
      }else{
        this.next.addNode(newNode);
      }
    }
    public boolean containsNode(String data){
      if(this.data.equals(data)){
        return true;
      }else{
        if(this.next!=null){
          return this.next.containsNode(data);
        }else{
          return false;
        }
      }
    }
    public String getNode(int index){
      if(Link.this.foot++==index){
        return this.data;
      }else{
        return this.next.getNode(index);
      }
    }
    public void setNode(int index,String data){
      if(Link.this.foot++==index){
        this.data=data;
      }else{
        this.next.setNode(index,data);
      }
    }
  }
  private Node root;
  private int count=0;
  private int foot=0;
  public void add(String data){
    Node newNode=new Node(data);
    if(this.root==null){
      this.root=newNode;
    }else{
      this.root.addNode(newNode);
    }
    count++;
  }
  public int size(){
    return this.count;
  }
  public boolean isEmpty(){
   return this.count==0;
  }
  public boolean contains(String data){
    if(data==null||this.root==null){
      return false;
    }else{
      return this.root.containsNode(data);
    }
  }
  public String get(int index){
    if(this.count<=index){
      return null;
    }
    this.foot=0;
    return this.root.getNode(index);
  }
  public void set(int index,String data){
    if(this.count<=index){
      return;
    }
    this.foot=0;
```

```
    this.root.setNode(index,data);
  }
}
public class LinkDemo{
  public static void main(String args[]){
  Link all=new Link();
  System.out.println(all.isEmpty());
  System.out.println(all.size());
  all.add("Hello");
  all.set(0,"xiaoyu");
  System.out.println(all.size());
  System.out.println(all.isEmpty());
  System.out.println(all.contains("Hello"));
  System.out.println(all.contains("We"));
  System.out.println(all.get(0));
  System.out.println(all.get(1));
  System.out.println(all.get(2));
  }
}
```

章节18 课时 74 03055_链表的定义与使用（删除链表数据）

```
class Link{
  private class Node{
    private String data;
    private Node next;
    public Node(String data){
      this.data=data;
    }
    public void addNode(Node newNode){
      if(this.next==null){
        this.next=newNode;
      }else{
        this.next.addNode(newNode);
      }
    }
    public boolean containsNode(String data){
      if(this.data.equals(data)){
        return true;
      }else{
        if(this.next!=null){
          return this.next.containsNode(data);
        }else{
          return false;
        }
      }
    }
    public String getNode(int index){
      if(Link.this.foot++==index){
        return this.data;
      }else{
        return this.next.getNode(index);
      }
    }
    public void setNode(int index,String data){
      if(Link.this.foot++==index){
        this.data=data;
      }else{
        this.next.setNode(index,data);
      }
    }
    public void removeNode(Node previous,String data){
      if(data.equals(this.data)){
        previous.next=this.next;
      }else{
        this.next.removeNode(this,data);
      }
    }
  }
  private Node root;
  private int count=0;
  private int foot=0;
  public void add(String data){
    Node newNode=new Node(data);
    if(this.root==null){
      this.root=newNode;
    }else{
      this.root.addNode(newNode);
    }
    count++;
  }
  public int size(){
    return this.count;
  }
  public boolean isEmpty(){
   return this.count==0;
  }
  public boolean contains(String data){
    if(data==null||this.root==null){
```

```
      return false;
    }else{
      return this.root.containsNode(data);
    }
  }
  public String get(int index){
    if(this.count<=index){
      return null;
    }
    this.foot=0;
    return this.root.getNode(index);
  }
  public void set(int index,String data){
    if(this.count<=index){
      return;
    }
    this.foot=0;
    this.root.setNode(index,data);
  }
  public void remove(String data){
    if(this.contains(data)){
      if(data.equals(this.root.data)){
        this.root=this.root.next;
      }else{
        this.root.next.removeNode(this.root,data);
      }
      this.count--;
    }
  }
}
public class LinkDemo{
  public static void main(String args[]){
  Link all=new Link();
  System.out.println(all.isEmpty());
  System.out.println(all.size());
  all.add("Hello");
  all.add("World");
  all.set(0,"xiaoyu");
  all.remove("World");
  System.out.println(all.size());
  System.out.println(all.isEmpty());
  System.out.println(all.contains("Hello"));
  System.out.println(all.contains("We"));
  System.out.println(all.get(0));
  System.out.println(all.get(1));
  System.out.println(all.get(2));

  }
}
```

章节18 课时 75 03056_链表的定义与使用（对象数组转换）

```
//链表数据变为对象数组取出是最为重要的功能
//如果要想定义对象数组（以类为例），可以采用如下的形式完成：
  //  动态初始化：
  //1、声明并开辟对象数组：类名称 对象数组名称[]=new 类名称[长度]
 // 2、分步完成：
     |-声明对象数组：类名称 对象数组名称[]=null;
     |-开辟对象数组：对象数组名称=new 类名称[长度];
class Link{
  private class Node{
    private String data;
    private Node next;
    public Node(String data){
      this.data=data;
    }
    public void addNode(Node newNode){
      if(this.next==null){
        this.next=newNode;
      }else{
        this.next.addNode(newNode);
      }
    }
    public boolean containsNode(String data){
      if(this.data.equals(data)){
        return true;
      }else{
        if(this.next!=null){
          return this.next.containsNode(data);
        }else{
          return false;
        }
      }
    }
    public String getNode(int index){
      if(Link.this.foot++==index){
        return this.data;
      }else{
        return this.next.getNode(index);
      }
```

```
    }
    public void setNode(int index,String data){
      if(Link.this.foot++==index){
        this.data=data;
      }else{
        this.next.setNode(index,data);
      }
    }
    public void removeNode(Node previous,String data){
      if(data.equals(this.data)){
        previous.next=this.next;
      }else{
        this.next.removeNode(this,data);
      }
    }
    public void toArrayNode(){
      Link.this.retArray[Link.this.foot++]=this.data;
      if(this.next!=null){//有后续元素
        this.next.toArrayNode();
      }
    }
  }
  private Node root;
  private int count=0;
  private int foot=0;
  private String [] retArray;//返回的数组
  public void add(String data){
    Node newNode=new Node(data);
    if(this.root==null){
      this.root=newNode;
    }else{
      this.root.addNode(newNode);
    }
    count++;
  }
  public int size(){
    return this.count;
  }
  public boolean isEmpty(){
   return this.count==0;
  }
  public boolean contains(String data){
    if(data==null||this.root==null){
      return false;
    }else{
      return this.root.containsNode(data);
    }
  }
  public String get(int index){
    if(this.count<=index){
      return null;
    }
    this.foot=0;
    return this.root.getNode(index);
  }
  public void set(int index,String data){
    if(this.count<=index){
      return;
    }
    this.foot=0;
    this.root.setNode(index,data);
  }
  public String [] toArray(){
    if(this.root==null){
      return null;
    }
    this.foot=0;//需要脚标控制
    this.retArray=new String[this.count];//根据保存内容开辟数组
    this.root.toArrayNode();//交给Node类处理
    return this.retArray;
  }
  public void remove(String data){
    if(this.contains(data)){
      if(data.equals(this.root.data)){
        this.root=this.root.next;
      }else{
        this.root.next.removeNode(this.root,data);
      }
      this.count--;
    }
  }
}
public class LinkDemo{
  public static void main(String args[]){
  Link all=new Link();
  System.out.println(all.isEmpty());
  System.out.println(all.size());
  all.add("Hello");
  all.add("World");
  all.add("nihao");
  all.set(0,"xiaoyu");
```

```
  all.remove("World");
  String [] data=all.toArray();
  for (int x=0;x<data.length;x++){
    System.out.println(data[x]);
  }
  System.out.println(all.size());
  System.out.println(all.isEmpty());
  System.out.println(all.contains("Hello"));
  System.out.println(all.contains("We"));
  System.out.println(all.get(0));
  System.out.println(all.get(1));
  System.out.println(all.get(2));
  }
}
```

章节18 课时 76 03057_链表的定义与使用（链表使用）

```
class Book{
  private String title;
  private double price;
  public Book(String title,double price){
    this.title=title;
    this.price=price;
  }
  public String getInfo(){
    return "图书名称："+this.title+"，价格："+this.price;
  }
  public boolean compare(Book book){
    if(this==book){
      return true;
    }
    if(book==null){
      return false;
    }
    if(this.title.equals(book.title)&&this.price==book.price){
      return true;
    }
    return false;
  }
}
class Link{
  private class Node{
    private Book data;
    private Node next;
    public Node(Book data){
      this.data=data;
    }
    public void addNode(Node newNode){
      if(this.next==null){
        this.next=newNode;
      }else{
        this.next.addNode(newNode);
      }
    }
    public boolean containsNode(Book data){
      if(data.compare(this.data)){
        return true;
      }else{
        if(this.next!=null){
          return this.next.containsNode(data);
        }else{
          return false;
        }
      }
    }
    public Book getNode(int index){
      if(Link.this.foot++==index){
        return this.data;
      }else{
        return this.next.getNode(index);
      }
    }
    public void setNode(int index,Book data){
      if(Link.this.foot++==index){
        this.data=data;
      }else{
        this.next.setNode(index,data);
      }
    }
    public void removeNode(Node previous,Book data){
      if(data.compare(this.data)){
        previous.next=this.next;
      }else{
        this.next.removeNode(this,data);
      }
    }
    public void toArrayNode(){
      Link.this.retArray[Link.this.foot++]=this.data;
      if(this.next!=null){//有后续元素
        this.next.toArrayNode();
```

```
    }
   }
  }
  private Node root;
  private int count=0;
  private int foot=0;
  private Book [] retArray;//返回的数组
  public void add(Book data){
    Node newNode=new Node(data);
    if(this.root==null){
      this.root=newNode;
    }else{
      this.root.addNode(newNode);
    }
    count++;
  }
  public int size(){
    return this.count;
  }
  public boolean isEmpty(){
   return this.count==0;
  }
  public boolean contains(Book data){
    if(data==null||this.root==null){
      return false;
    }else{
      return this.root.containsNode(data);
    }
  }
  public Book get(int index){
    if(this.count<=index){
      return null;
    }
    this.foot=0;
    return this.root.getNode(index);
  }
  public void set(int index,Book data){
    if(this.count<=index){
      return;
    }
    this.foot=0;
    this.root.setNode(index,data);
  }
  public Book [] toArray(){
    if(this.root==null){
      return null;
    }
    this.foot=0;//需要脚标控制
    this.retArray=new Book[this.count];//根据保存内容开辟数组
    this.root.toArrayNode();//交给Node类处理
    return this.retArray;
  }
  public void remove(Book data){
    if(this.contains(data)){
      if(data.compare(this.root.data)){
        this.root=this.root.next;
      }else{
        this.root.next.removeNode(this.root,data);
      }
      this.count--;
    }
  }
}
public class LinkDemo{
  public static void main(String args[]){
  Link all=new Link();
    all.add(new Book("Java开发",79.8));
    all.add(new Book("Jsp开发",69.8));
    all.add(new Book("Oracle开发",89.8));
    System.out.println("保存书的个数："+all.size());
    System.out.println(all.contains(new Book("Java开发",79.8)));
    all.remove(new Book("Oracle开发",89.8));
    Book [] books=all.toArray();
    for(int x=0;x<books.length;x++){
      System.out.println(books[x].getInfo());
    }
  }
}
```

章节18 课时 77 03058_链表的定义与使用（在映射中使用链表）

```
class Province{
  private int pid;
  private String name;
  private Link cities=new Link();
  public Province(int pid,String name){
  this.pid=pid;
  this.name=name;
  }
  public boolean compare(Province province){
    if(this==province){
```

```
      return true;
    }
    if(province==null){
      return false;
    }
    if(this.pid==province.pid&&this.name.equals(province.name)){
      return true;
    }
    return false;
  }
  public Link getCities(){
    return this.cities;
  }
  public String getInfo(){
    return "省份编号：" +this.pid+"，名称：" +this.name;
  }

}
class City{
  private int cid;
  private String name;
  private Province province;
  public  City(int cid,String name){
    this.cid=cid;
    this.name=name;
  }
  public boolean compare(City city){
    if(this==city){
      return true;
    }
    if(city==null){
      return false;
    }
    if(this.cid==city.cid
      &&this.name.equals(city.name)
      &&this.province.compare(city.province)){
      return true;
    }
    return false;
  }
  public void setProvince(Province province){
    this.province=province;
  }
  public Province getProvince(){
    return this.province;
  }
  public String getInfo(){
    return "城市编号：" +this.cid+"，名称"+this.name;
  }
}
class Link{
  private class Node{
    private City data;
    private Node next;
    public Node(City data){
      this.data=data;
    }
    public void addNode(Node newNode){
      if(this.next==null){
        this.next=newNode;
      }else{
        this.next.addNode(newNode);
      }
    }
    public boolean containsNode(City data){
      if(data.compare(this.data)){
        return true;
      }else{
        if(this.next!=null){
          return this.next.containsNode(data);
        }else{
          return false;
        }
      }
    }
    public City getNode(int index){
      if(Link.this.foot++==index){
        return this.data;
      }else{
        return this.next.getNode(index);
      }
    }
    public void setNode(int index,City data){
      if(Link.this.foot++==index){
        this.data=data;
      }else{
        this.next.setNode(index,data);
      }
    }
    public void removeNode(Node previous,City data){
      if(data.compare(this.data)){
```

```
        previous.next=this.next;
      }else{
        this.next.removeNode(this,data);
      }
    }
    public void toArrayNode(){
      Link.this.retArray[Link.this.foot++]=this.data;
      if(this.next!=null){//有后续元素
        this.next.toArrayNode();
      }
    }
  }
  private Node root;
  private int count=0;
  private int foot=0;
  private City [] retArray;//返回的数组
  public void add(City data){
    Node newNode=new Node(data);
    if(this.root==null){
      this.root=newNode;
    }else{
      this.root.addNode(newNode);
    }
    count++;
  }
  public int size(){
    return this.count;
  }
  public boolean isEmpty(){
   return this.count==0;
  }
  public boolean contains(City data){
    if(data==null||this.root==null){
      return false;
    }else{
      return this.root.containsNode(data);
    }
  }
  public City get(int index){
    if(this.count<=index){
      return null;
    }
    this.foot=0;
    return this.root.getNode(index);
  }
  public void set(int index,City data){
    if(this.count<=index){
      return;
    }
    this.foot=0;
    this.root.setNode(index,data);
  }
  public City [] toArray(){
    if(this.root==null){
      return null;
    }
    this.foot=0;//需要脚标控制
    this.retArray=new City[this.count];//根据保存内容开辟数组
    this.root.toArrayNode();//交给Node类处理
    return this.retArray;
  }
  public void remove(City data){
    if(this.contains(data)){
      if(data.compare(this.root.data)){
        this.root=this.root.next;
      }else{
        this.root.next.removeNode(this.root,data);
      }
      this.count--;
    }
  }
}
public class LinkDemo{
  public static void main(String args[]){
  Link all=new Link();
   //设置关系数据
    //1、先准备好各自独立的对象；
    Province pro=new Province(1,"河北省");
    City c1=new City(1001,"唐山");
    City c2=new City(1002,"秦皇岛");
    City c3=new City(1003,"石家庄");
    c1.setProvince(pro);
    c2.setProvince(pro);
    c3.setProvince(pro);
    pro.getCities().add(c1);
    pro.getCities().add(c2);
    pro.getCities().add(c3);
    //取出关系
    System.out.println(pro.getInfo());
    System.out.println("拥有的城市数量："+pro.getCities().size());
    pro.getCities().remove(c1);
```

```
    City c[]=pro.getCities().toArray();
    for(int x=0;x<c.length;x++){
      System.out.println(c[x].getInfo());
    }
  }
}
```

//总结：
在95%的操作情况下，链表里面有2个功能是最常用的，一个是增加，另一个是取得全部内容。

No	方法名称	类型	描述
1	**private void add(数据类型,变量)**	普通	向链表之中增加新
2	public int size()	普通	取得链表中保存的元素个数
3	public boolean isEmpty()	普通	判断是否是空链表(size()==0)
4	public boolean contains(数据类型,变量)	普通	判断一个数据是否存在
5	public 数据类型 get(int index)	普通	根据索引取得数据
6	public void set(int index,数据类型 变量)	普通	使用新的内容替换掉指定索引的旧内容
7	public void remove(数据类型 变量)	普通	删除指定数据，如果是对象，则要进行对象比较
8	**public 数据类型 [] toArray()**	普通	将链表以对象数组的形式返回

章节19 课时 78 03059_继承性（继承问题的引出）

```
class Student{
  private String name;
  private int age;
  private String school;
  public void setName(String name){
    this.name=name;
  }
  public void setAge(int age){
    this.age=age;
  }
  public void setSchool(String school){
    this.school=school;
  }
  public String getName(){
    return this.name;
  }
  public int getAge(){
    return this.age;
  }
  public String getSchool(){
    return this.school;
  }
}
class Person{
  private String name;
  private int age;
  public void setName(String name){
    this.name=name;
  }
  public void setAge(int age){
    this.age=age;
  }
  public String getName(){
    return this.name;
  }
  public int getAge(){
    return this.age;
  }
}
public class TestDemo{
  public static void main(String args[]){

  }
}
```

章节19 课时 79 03060_继承性（继承的实现）

```
class Person{
  private String name;
  private int age;
  public void setName(String name){
    this.name=name;
  }
  public void setAge(int age){
    this.age=age;
  }
```

```
  public String getName(){
    return this.name;
  }
  public int getAge(){
    return this.age;
  }
}
class Student extends Person{//继承了Person父类
  private String school;
  public void setSchool(String school){
    this.school=school;
  }
  public String getSchool(){
    return this.school;
  }
}
public class TestDemo{
  public static void main(String args[]){
    Student stu=new Student();
    stu.setName("阿尤");
    stu.setAge(24);
    stu.setSchool("家里蹲大学");
    System.out.println("姓名："+stu.getName()+"，年龄："+stu.getAge()+"，学校："+stu.getSchool());
  }
}
```

章节19 课时 80 03061_继承性（继承的限制）

```
//Java不允许多重继承，可以多层继承，开发角度而言，不要超过3层。
//子类在继承父类的时候严格来讲会继承父类的所有操作，但是对于所有的私有操作属于隐式继承，而所有的非私有操作属于显式继承。

class A{
  private String msg;
  public void setMsg(String msg){
    this.msg=msg;
  }
  public String getMsg(){
    return this.msg;
  }
}
class B extends A{
  //public void fun(){
  //  System.out.println(msg);//因为msg为私有属性，属于隐式继承，不能被操作，编译无法通过。
  //}
}
public class TestDemo{
  public static void main(String args[]){
    B b=new B();
    b.setMsg("Hello");
    System.out.println(b.getMsg());
  }
}
```

```
class A{
  public A(){
    System.out.println("A-A类的构造方法");
  }
}
class B extends A{
  public B(){
    super ();//父类中有无参构造是，加与不加无区别。
    System.out.println("B-B类的构造方法");
  }
}
public class TestDemo{
  public static void main(String args[]){
    new B();//此时并没有任何的操作代码，但发现，在实例化子类对象前先去实例化了父类对象以及调用了父类的无参构造方法。那么此时对于子类构造而言，就相当于隐藏了一个"super()"。
  }
}

//如果此时父类中没有无参构造方法了，那么就必须使用super()明确调用父类的有参构造方法。
class A{
  public A(String title){
    System.out.println("A-A类的构造方法");
  }
}
class B extends A{
  public B(String title){
    super(title);
    System.out.println("B-B类的构造方法");
  }
}
```

```
public class TestDemo{
  public static void main(String args[]){
    new B("Hello");
  }
}


//通过观察super()主要是由子类调用父类中的构造方法，那么这行语句一定要放在子类构造方法的首行。这一点和this()是类似的。那么这两个同时出现呢？
//通过代码验证：super()与this()不能同时出现，不管子类怎么折腾，它都永恒有一个存在的前提：子类对象的构造调用前一定先执行父类构造，为父类的对象初始化后，才轮到子类对象初始化。
```

章节20 课时 81 03062_覆写（方法覆写）

```
class A{
  public void fun(){
    System.out.println("A-A类的fun()方法");
  }
}
class B extends A{

}
public class TestDemo{
  public static void main(String args[]){
   B b=new B();
   b.fun();
  }
}

//下面开始覆写
class A{
  public void fun(){
    System.out.println("A-A类的fun()方法");
  }
}
class B extends A{
  public void fun(){
    System.out.println("B-B类的fun()方法");
  }
}
public class TestDemo{
  public static void main(String args[]){
   B b=new B();
   b.fun();
  }
}


class A{
  public void fun(){
    System.out.println("A-A类的fun方法");
  }
}
class B extends A{
  public void fun(){
    System.out.println("B-B类的fun方法");
  }
}
class C extends A{
  public void fun(){
    System.out.println("C-C类的fun方法");
  }
}
public class TestDemo{
  public static void main(String args[]){
   B b=new B();
   b.fun();
   C c=new C();
   c.fun();
  }
}

class A{
  public void fun(){
    print();
  }
  private void print(){
    System.out.println("大家下午辛苦了，瞌睡的同学请站到教室后面去。");
  }
}
class B extends A{
  public void print(){
    System.out.println("姜同学自己罚站了");
  }
}
public class TestDemo{
  public static void main(String args[]){
    B b=new B();
```

```
    b.fun();
    b.print();
  }
}
```
//这个时候发现子类中根本就没有复写print()方法，也就是说如果使用了private声明，那么这个方法对子类而言是不可见的，就算子类顶一个了一个与之完全相同的符合于覆写要求的方法，那么也不能够发生覆写。实际上就相当于子类自

```
class A{
  public void print(){
    System.out.println("大家下午辛苦了，瞌睡的同学请站到教室后面去。");
  }
}
class B extends A{
  public void print(){
    //this.print();//出现递归调用死循环。
    super.print();
    System.out.println("姜同学自己罚站了");
  }
}
public class TestDemo{
  public static void main(String args[]){
    B b=new B();
    b.print();
  }
}
```
//使用this.方法()会首先查找本类中是否存在要有调用的方法名称，如果存在则直接调用，如果不存在则查找父类中是否具备有次方法，如果有就调用，如果没有，则会发生编译时的错误提示；
//使用super.方法()，明确的表示调用的不是子类方法（不查找子类中是否存在有此方法，而直接调用父类中的此方法。）

No	区别	重载	覆写
1	英文单词	Overloading	Override
2	发生范围	发生在一个类里	发生在继承关系中
3	定义	方法名称相同，参数的类型及个数相同，返回值类型可以不同但不建议。	方法名称相同，参数的类型及个数相同，方法返回值相同
4	权限	没有权限的限制	被覆写的方法不能拥有比父类更严格的访问控制权限。

章节20 课时 82 03063_覆写（属性覆盖）

```
class A{
  String info="Hello";
}
class B extends A{
  int info=100;
  public void print(){
    System.out.println(info);
    System.out.println(super.info);
  }
}
public class TestDemo{
  public static void main(String args[]){
    B b=new B();
    b.print();
  }
}
```

NO	区别	this	super
1	功能	调用本类构造、本类方法、本类属性	子类调用父类构造、父类方法、父类属性
2	形式	先查找本类中是否存在有指定的调用结构，如果有则直接调用，如果没有则调用父类定义。	不查找子类，直接调用父类操作
3	特殊	表示本类的当前对象	不能单独使用

//1、在以后的开发中，强烈建议在本类或者父类中加上"this."或者"super."，这样好区分。
//2、如果子类中要用父类指定的方法，但是发现父类的方法实现不能满足子类要求的时候就要使用覆写来完善子类的功能，同时保留父类的方法名称。
//3、被子类覆写的方法不能拥有比父类更严格访问控制权限。