# 对象转型

有句话说的好: **就类似于一个本来就是儿子角色的人, 去装了一会儿父亲, 这个角色是可以在未来转换回去的, 但是一个父亲, 永远不可能去装儿子的**.

引用只能拿到自己的东西.

# abstract和interface

当描述一组方法的时候使用接口  当描述一个虚拟的物体的时候使用抽象类,

|  |
| --- |
| public abstract class Employee {  protected String firstName, lastName;  protected int salary, bonus;   public Employee(String firstName, String lastName, int salary) {  this.firstName = firstName;  this.lastName = lastName;  this.salary = salary;  }   public Employee() {   }  省略get set  abstract void setBonus(); } |
| public interface Hobby {  void playBall();   void game(); } |
| public class TopVip extends Employee implements Hobby {  @Override  void setBonus() {  bonus = getSalary() \* 30 / 100;  }   public TopVip(String firstName, String lastName, int salary) {  super(firstName, lastName, salary);  setBonus();  }   public TopVip() {   }   @Override  public String toString() {  return "TopVip{" +  "firstName='" + firstName + '**\'**' +  ", lastName='" + lastName + '**\'**' +  ", salary=" + salary +  ", bonus=" + bonus +  '}';  }   @Override  public void playBall() {  System.out.println("can play ball");  }   @Override  public void game() {  System.out.println("can play game");  } } |
| public class TopPt extends Employee implements Hobby {  @Override  void setBonus() {  bonus = getSalary() \* 10 / 100;  }   public TopPt(String firstName, String lastName, int salary) {  super(firstName, lastName, salary);  setBonus();  }   @Override  public String toString() {  return "TopPt{" +  "firstName='" + firstName + '**\'**' +  ", lastName='" + lastName + '**\'**' +  ", salary=" + salary +  ", bonus=" + bonus +  '}';  }   public TopPt() {  }    @Override  public void playBall() {  System.out.println("can play ball");  }   @Override  public void game() {   } } |
| public class MainTest {   @Test  public void test1() { // 创建一个员工  Employee ee; // 创建一个vip员工  new TopVip("yang", "shisheng", 100); // pt员工  ee = new TopPt("yang", "xingxing", 100); // 输出基本信息:  System.out.println("输出基本信息" + ee);  Hobby hobby = (Hobby) ee; //输出爱好信息  hobby.game();  hobby.playBall();  }  } |

# 泛型

|  |
| --- |
| public class Demo {  public static void main(String[] args) {  // 实例化泛型类  Point<Integer, Integer> p1 = new Point<Integer, Integer>();  p1.setX(10);  p1.setY(20);  int x = p1.getX();  int y = p1.getY();  System.out.println("This point is：" + x + ", " + y);   Point<Double, String> p2 = new Point<Double, String>();  p2.setX(25.4);  p2.setY("东京180度");  double m = p2.getX();  String n = p2.getY();  System.out.println("This point is：" + m + ", " + n);  } }  // 定义泛型类 class Point<T1, T2> {  T1 x;  T2 y;   public T1 getX() {  return x;  }   public void setX(T1 x) {  this.x = x;  }   public T2 getY() {  return y;  }  public void setY(T2 y) {  this.y = y;  } } |

泛型三种：  
          [1]ArrayList<T> al=new ArrayList<T>();指定集合元素只能是T类型  
          [2]ArrayList<?> al=new ArrayList<?>();集合元素可以是任意类型，这种没有意义，一般是方法中，只是为了说明用法  
          [3]ArrayList<? extends E> al=new ArrayList<? extends E>();  
            泛型的限定：  
               ? extends E:接收E类型或者E的子类型。  
               ？super E:接收E类型或者E的父类型。