# Java基础

zcx

## 一、面向对象编程

1. **继承**

- 让重复的代码总合成一个类，作为父类，让其他子类从父类中读取数据并进一步扩展，就是继承，减少重复代码的书写
- **extends** 关键字：public class B extends A{}
  - A为父类，B为子类
  - 子类能继承父类的非私有成员
  - 子类的对象是由子类和父类共同完成的
  - ```
    public class people { 父类
    private String name;
    private char sex;
    public String getName() {
    return name;
    }
    public char getSex() {
    return sex;
    }
    public void setName(String name) {
    this.name = name;
    }
    public void setSex(char sex) {
    this.sex = sex;
    }
    }
    public class teacher extends people{ 老师类
    private String skill;
    public void setSkill(String skill) {
    this.skill = skill;
    }
    public String getSkill() {
    return skill;
    }
    ```
    ```

```
}
public class consultant extends people{ 咨询师类
private int number;
public int getNumber() {
return number;
}
public void setNumber(int number) {
this.number = number;
}
}
public class test { 主函数类
public static void main(String[] args) {
teacher t = new teacher(); 老师类对象声明
t.setName("CXZ"); 继承了父类的公有方法
t.setSex('男'); 同上
t.setSkill("C++"); 老师类中的方法
consultant c = new consultant(); 咨询师类对象声明
c.setName("CXZsss"); 继承了父类的公有方法
c.setSex('女'); 同上
c.setNumber(2); 咨询师类中的方法
}
}
```

- **权限修饰符**
  - 用来限制类中的成员(成员变量，成员方法，构造器)能够被访问的范围，如下的范围依次扩大
  - **private**:只能本类
  - **不加任何修饰符**：本类或同个包中的类
  - **protected**：本类，同一个包的类，子孙类(可以是其他包的子类)
  - **public**：任何位置
- **继承的特点**
  - **单继承**：Java是单继承模式，一个类只能继承一个直接父类
  - **多层继承**：Java支持多层继承，即子类有父类，父类有父类的父类，**但是不能直接跨级调用，需要父类中转一下**，但是Java不支持多继承，即子类不能有多个父类，**因为如果多个类中有相同的方法名，但函数构成不同，子类同时继承这些类，方法则会混乱**
  - **祖宗类**：Java中的所有类都是Object类的子类，一个类要么直接继承Object类，要么间接继承Object类
    - 按ctrl，鼠标左键点击Object就会看到他的源码
    - ```
      public class test extends Object
      ```
  - **就近原则**：优先访问自己类中的，没有才访问父类中的
    - ```
      public class test2 {
      public static void main(String[] args) {
      ```

```
zi z = new zi();
z.print();
}
}
class fu {
String name="fu"; 父类对象的name
}
class zi extends fu{
String name="zi"; 子类对象的name
public void print(){
String name = "lalalalal"; 子类的方法的name
System.out.println(name); 输出lalalalal
System.out.println(this.name); 输出zi
System.out.println(super.name); 输出fu
}即，要想在子类访问父类中两者重名的成员，就需要用super关键字访问
}但是不能用super.super调用祖父类的重名成员，若父类没有重写该方法，则super即可调用祖父类的重名成员，要是父类重写了还想要在子类调用祖父类的重名成员，则需要在父类中提供访问途径
```

- **方法重写**
  - 当子类觉得父类中的某个方法不好用，或者无法满足自己的需求，子类可以重写一个方法名称、参数列表一样的方法，去覆盖父类的这个方法。
  - ```
    public class test {
    public static void main(String[] args) {
    cat c = new cat();
    c.cry();
    }
    }
    class animal{
    public void cry(){
    System.out.println("动物会叫");
    }
    }
    class cat extends animal{
    @Override //方法重写的校验注解，告诉编译器，我重写这个方法，方法名要是不一样就会报错=
    public void cry(){
    System.out.println("喵喵喵");
    }
    }
    ```
    ```

- 注意事项：子类重写父类方法时，访问权限必须大于等于父类该方法的权限。重写的方法返回值类型，必须与被重写的返回值类型一样，或者范围更小。私有方法，静态方法不能被重写。
- 方法重写的常见应用场景:
  - 子类重写Object类的toString()方法，以便返回对象的内容
  - ***可以鼠标右键，generate，找到toString即可自动书写***
  - ```
    public class test2 {
    public static void main(String[] args) {
    //子类重写Object类的toString()方法，以便返回对象的内容
    student s = new student("CXZ",18,'男');
    System.out.println(s); //默认调用的是s.toString()方法，返回对象s的地址值
    //输出对象的地址没有意思，想看的是对象的内容信息，故需要重写toString()方法，
    // 以便输出对象时默认就近重写的toString()方法,来输出对象内容。
    }
    }
    class student{
    private String name;
    private int age;
    private char sex;
    public student(String name,int age,char sex){
    this.name = name;
    this.age = age;
    this.sex = sex;
    }
    public student(){}
    public String getName() {
    return name;
    }
    public int getAge() {
    return age;
    }
    public char getSex() {
    return sex;
    }
    public void setName(String name) {
    this.name = name;
    }
    public void setAge(int age) {
    this.age = age;
    ```
    ```

```
}
public void setSex(char sex) {
this.sex = sex;
}
@Override
public String toString() { //重写后就是输出对象内容了
return "姓名："+name+"，年龄："+age+"，性别："+sex;
}
}
```

- **子类构造器的特点**
  - 子类的全部构造器，都会先调用父类的构造器，在执行自己
  - 默认情况下，子类的全部构造器第一行代码都是super()，写不写都有，他会调用父类的无参数构造器。
  - 如果父类没有无参数构造器，则我们必须在子类构造器的第一行手写super(...)，指定去调用父类的有参数构造器
  - ```
    public class test {
    public static void main(String[] args) {
    zi z = new zi();
    }
    }
    class fu{
    public fu(){
    System.out.println("父类无参构造");
    }
    }
    class zi extends fu{
    public zi(){
    System.out.println("子类无参构造");
    }
    }最后输出第一行为父类无参构造，第二行为子类无参构造
    ```
  - ```
    class fu{
    private String name;
    private char sex;
    public fu(){
    }
    public fu(String name,char sex){父类的有参构造器
    this.name = name;
    this.sex = sex;
    }
    }
    ```

```
class zi extends fu{
private String skill;
public zi(String name,char sex,String skill){ 子类的有参构造器
super(name,sex);子类继承自父类的数据也能完成初始化赋值
this.skill = skill;
}
}
```

- **this(...)调用兄弟构造器**
  - 在构造器中调用本类的其他构造器
  - ```
    package construct;
    public class student {
    private String name;
    private int age;
    private String sex;
    private String schoolname;
    public student(){
    System.out.println("无参构造方法");
    }
    public student(String name,int age,String sex,String schoolname){
    this.name = name;
    this.age = age;
    this.sex = sex;
    this.schoolname = schoolname;
    }
    public student(String name,int age,String sex){
    this(name, age, sex, "CXZ");
    }我想要默认设置第四个变量为固定值，但是还想正常赋值前几个变量，就可以这么操作简化代码
    public String getName() {
    return name;
    }
    public int getAge() {
    return age;
    }
    public String getSex() {
    return sex;
    }
    public String getSchoolname() {
    return schoolname;
    }
    ```
    ```

```
public void setName(String name) {
this.name = name;
}
public void setAge(int age) {
this.age = age;
}
public void setSex(String sex) {
this.sex = sex;
}
public void setSchoolname(String schoolname) {
this.schoolname = schoolname;
}
@Override
public String toString() {
return "student{" +
"name='" + name + '\'' +
", age=" + age +
", sex='" + sex + '\'' +
", schoolname='" + schoolname + '\'' +
'}';
}
}
public class test3 {
public static void main(String[] args) {
student s1 = new student("cxz",19,"男");
System.out.println(s1);
student s2 = new student("cxz1",19,"男");
System.out.println(s2);
}
}
```

- ***注意事项：super(...)和this(...)都必须写在构造器的第一行，并且两者不能同时出现***

2. **多态**

- **认识多态**
  - 在继承/实现情况下的一种现象，表现为现象多态(同样都是人，但身份不同)、行为多态(都是动物，乌龟跑的慢，老虎跑的快)
  - 多态的前提：有继承/实现关系，存在父类引用子类对象，存在方法重写
  - ```
    public class test {
    public static void main(String[] args) {
    //对象多态
    ```
    ```

```
animal a = new wolf(); //父类引用指向子类对象
animal b = new tortoise(); //父类范围更大，这是子类赋给父类引用
//行为多态(成员方法)
a.run(); //编译看左边，运行看右边
b.run(); //编译看左边animal有没有run方法，运行看右边子类有没有run方法
}
}但是成员变量不同，编译看左边，运行也看左边，都是动物类的成员变量(因为没有成员变量的多态，变量是不会找真正对象的)
public class animal {
public void run(){
System.out.println("动物都会跑");
}
}
public class wolf extends animal{
public void run(){
System.out.println("狼跑的快");
}
}
public class tortoise extends animal{
public void run(){
System.out.println("乌龟跑得慢");
}
}
```

- **多态的好处**
  - 在多态形式下，右边对象是解耦合的，会更利于扩展和维护
  - 解耦合是指：降低模块间的依赖，即使这个模块出问题，我可以无缝换下一个模块保证整体代码继续运行，而不需要大量修改原有旧代码。
  - ```
    public class test {
    public static void main(String[] args) {
    //解耦合
    animal a = new wolf();
    a.run();
    }
    }那么现在我不想用wolf了，我想用tortoise，那么直接把wolf换掉，而不需要修改其他代码
    public class test {
    public static void main(String[] args) {
    //解耦合
    animal a = new tortoise();
    a.run();
    ```
    ```

```
}
}
```

- 定义方法时，使用父类类型的形参，可以接收一切子类对象，扩展性更强，更便利
- ```
  public class test {
  public static void main(String[] args) {
  //解耦合
  animal a = new tortoise();
  a.run();
  wolf w = new wolf();
  go(w); 一种是狼类型的变量
  tortoise t = new tortoise();
  go(t); 一种是乌龟类型的变量
  }
  public static void go(animal a){ 用的是animal声明的对象，狼和乌龟都能传入，这里就是多态
  System.out.println("go go go...");
  a.run();
  }如果不是方法不是animal a 而是wolf w的话，那么就无法传入其他动物的对象
  }
  ```
  - 但是多态会产生一个问题：**多态下无法调用子类的独有功能**，因为编译看左边，左边的父类没有子类独有的功能

- **多态下的类型转换**
  - 自动类型转换：父类 变量名 = new 子类();
  - *强制类型转换：子类 变量名 = (子类)父类变量*
  - ```
    public class test {
    public static void main(String[] args) {
    //解耦合
    animal a = new tortoise();
    a.run();
    //强制类型转换
    tortoise t1 = (tortoise) a;
    t1.eat(); 这是乌龟类的独有功能
    }这就把父类的变量名a强制转换为子类的对象名t1了，然后t1就可以调用子类的独有功能
    }
    ```
    - 注意事项：存在继承/实现关系就可以在编译阶段进行强转，编译阶段不会报错，但是运行时，如果发现对象的真实类型与强转后的类型不同，就会报类型转换异常的错误
  - ```
    animal a = new tortoise();
    wolf w = (wolf) a;
    ```
    ```

> ***编译时不会出错，因为存在wolf和a对象存在继承关系，但是，运行时一看，把a对象的真实类型是乌龟，强转成狼了，就会报错***

- 强转前，使用 ***instanceof*** 关键字，判断当前对象的真实类型，再进行强转
- > a instanceof wolf 用if else的判断条件，比如在方法参数是多态时，根据不同的传入的类型，进一步实现下一步不同的操作

- ```
  wolf w = new wolf();
  go(w);
  tortoise t = new tortoise();
  go(t);
  public static void go(animal a){
  System.out.println("go go go...");
  a.run();
  if(a instanceof tortoise){ 判断类型为tortoise是真
  tortoise t = (tortoise) a; 强转为tortoise
  t.eat(); 执行乌龟的独有功能
  }else if(a instanceof wolf){ 判断类型为wolf是真
  wolf w = (wolf) a; 强转为wolf
  w.jump(); 执行狼的独有功能
  }
  }
  ```

3. **综合实战**

- 加油站支付小模块：某加油站为了吸引更多的车主，推出了如下活动，车主可以办理金卡和银卡。卡片信息包括：车牌号码、车主姓名、电话号码、卡片余额。金卡办理时入存金额必须大于等于5000元，银卡则大于等于2000元，金卡支付时享有8折优惠，银卡则是9折，金卡消费满200可以提供打印免费洗车票的服务。需求：请使用面向对象编程，完成加油站支付机的存款和消费程序
- ```
  import java.util.Scanner;
  public class test { 这是主类
  public static void main(String[] args){
  goldcard gc = new goldcard("CXZ", "CXZ", "123456", 5000); 声明金卡类对象
  silvercard sc = new silvercard("CXZ1", "CXZ1", "654321",2000); 声明银卡类对象
  pay(gc); 调用pay方法，即加油站支付机操作
  pay(sc);
  }
  public static void pay(card c){ 支付方法
  System.out.println("请刷卡，请输入您消费的金额：");
  Scanner scc = new Scanner(System.in);
  double money = scc.nextDouble();
  if(c instanceof goldcard){ 判断对象类型
  ```

```
goldcard gc = (goldcard)c; 强转为金卡类对象
gc.consume(money);
}else if(c instanceof silvercard){
silvercard sc = (silvercard)c; 强转为银卡类对象
sc.consume(money);
}
}
}
```

- ```
  import lombok.Data;
  import lombok.AllArgsConstructor;
  import lombok.NoArgsConstructor; //有这些就不用自己写那些方法了，编译时一键导入了
  @Data // getter setter equals hashCode toString方法
  @AllArgsConstructor // 全参构造器
  @NoArgsConstructor // 无参构造器
  public class card { 这是卡类，父类
  private String cid;
  private String name;
  private String phone;
  private double balance;
  //预存金额
  public void save(double money){
  this.balance += money;
  }
  //消费金额
  public void consume(double money){
  this.balance -= money;
  }
  }
  ```

- ```
  public class goldcard extends card{ 这是金卡类，子类
  public goldcard(String cxz, String cxz1, String number, int i) {
  super(cxz, cxz1, number, i); 调用父类构造器
  }
  public void consume(double money) {
  System.out.println("使用金卡消费,您当前消费："+money);
  System.out.println("优惠后价格："+money0.8);
  if(getBalance()<money0.8){ 判断余额是否充足
  System.out.println("余额不足，请充值");
  }else{
  setBalance(getBalance()-money0.8); //更新后的余额
  System.out.println("当前余额："+getBalance());
  ```

```
if(money0.8>=200){ 判断金额是否满200
printticket();
}else{
System.out.println("金卡会员消费不满200，无法打印洗车票");
}
}
}
public void printticket(){
System.out.println("金卡会员消费满200，请打印洗车票");
}
}
```

- ```
  public class silvercard extends card{ 这是银卡类，子类
  public silvercard(String cxz, String cxz1, String number, int i) {
  super(cxz, cxz1, number, i); 调用父类构造器
  }
  public void consume(double money) {
  System.out.println("使用银卡消费,您当前消费："+money);
  System.out.println("优惠后价格："+money0.9);
  if(getBalance()<money0.9){ 判断余额是否充足
  System.out.println("余额不足，请充值");
  }else{
  setBalance(getBalance()-money*0.9); //更新后的余额
  System.out.println("当前余额："+getBalance());
  }
  }
  }
  ```

- ```
  最终输出结果：
  请刷卡，请输入您消费的金额：
  300
  使用金卡消费,您当前消费：300.0
  优惠后价格：240.0
  当前余额：4760.0
  金卡会员消费满200，请打印洗车票
  请刷卡，请输入您消费的金额：
  200
  使用银卡消费,您当前消费：200.0
  优惠后价格：180.0
  当前余额：1820.0
  ```