**JAVA基础**

# 面向对象

三大特征：封装 继承 多态

封装encapsulation

* 封装包含两方面含义：
  + 合理隐藏。
  + 合理暴隐。

是把客观事物封装成抽象的类，并且类可以把自己的数据和方法只让可信的类或者对象操作，对不可信的进行信息隐藏，隐藏对象的内部实现细节

封装（Encapsulation）是面向对象方法的重要原则，就是把对象的属性和操作（或服务）结合为一个独立的整体，并尽可能隐藏对象的内部实现细节

好处：

① 对象的数据封装特性彻底消除了传统结构方法中数据与操作分离所带来的种种问题，提高了程序的可复用性和可维护性，降低了程序员保持数据与操作内容的负担。

②对象的数据封装特性还可以把对象的私有数据和公共数据分离开，保护了私有数据，减少了可能的模块间干扰，达到降低程序复杂性、提高可控性的目的。

使用访问权限控制符：

* private 私有的。在同一个类里能被访问。
* default 默认的。包访问权限 同包下可以访问，前面不加修饰符时就是default
* protected 受保护的。子类中也能访问/包访问权限
* public 公共的。在任何地方都可以访问
* 既可以修饰属性，也可以修饰方法
* 访问修饰符可以体现合理隐藏与合理暴露

|  |
| --- |
| **public** **class** Student {  // 在同一个类里(私有属性所在的类)能被访问，  // 虽然不能在其他类中直接访问，但是我们可以为其提供使用public  // 修饰的get、set方法，对其进行设置值或取其值  **private** String name;  // 在任何地方都可以访问  **public** String address;  // 没有修饰符，就是默认访问权限，默认访问权限也称为包访问权限  // 即同包下可以访问，不同包下不能访问，如果想访问的话，添加使用  // public修饰的get、set方法即可  **int** age;  // 受保护的，包访问权限 || 子类可以访问  **protected** String gender;      **public** Student() {    }  /\*\*  \* 构造器，可以用来初始化私有属性  \* **@param** name  \* **@param** address  \* **@param** age  \* **@param** gender  \*/  **public** Student(String name, String address, **int** age, String gender) {  **super**();  **this**.name = name;  **this**.address = address;  **this**.age = age;  **this**.gender = gender;  }  **public** String getName() {  **return** **this**.name;  }    **public** **void** setName(String name) {  **this**.name = name;  }    **public** **int** getAge() {  **return** age;  }  **public** **void** setAge(**int** age) {  **this**.age = age;  }    /\*\*  \* 黄色波浪线提示该方法没有被使用过，因为其使用私有private修饰  \* 只能在当前类中被调用，如果没有调用，就会给出这个提示  \*/  **private** **void** test() {  System.***out***.println("test");  }  } |
| **public** **static** **void** main(String[] args) {  Student student = **new** Student();    System.***out***.println(student.address);  // name是私有属性，不能再这里访问，想访问的话，可一个通过  // 其get、set方法访问，如下get、set方法  // System.out.println(student.name);  student.setName("张三");  System.***out***.println(student.getName());    System.***out***.println(student.age);  System.***out***.println(student.gender);  } |
| **Package com.congnizant.java（不同包）**  **public** **static** **void** main(String[] args) {  Student student = **new** Student();    System.***out***.println(student.address);  // name是私有属性，不能再这里访问，想访问的话，可一个通过  // 其get、set方法访问  // System.out.println(student.name);  student.setName("张三");  System.***out***.println(student.getName());    // System.out.println(student.age);  System.***out***.println(student.getAge());    // System.out.println(student.gender);  } |
| **import** com.cognizant.Student;  /\*\*  \* Student父类  \* SubStudent子类  \* **@author** tzhang  \*  \*/  **public** **class** SubStudent **extends** Student {  **public** **void** test() {  // 从父类中继承这个属性,通过super来访问父类的属性  System.***out***.println(**super**.gender);  }  } |

继承Inheritance/extends

* Java通过关键字extends来实现，实现继承的类称为子类，被继承的类称为基类、超类、父类。父类是大类，子类是小类。
* Java的继承是单继承，每个子类最多只有一个直接父类。
* Java继承是使用已存在的类的定义作为基础建立新类的技术

继承是一种能力：它可以使用现有类的所有功能，并在无需重新编写原来的类的情况下对这些功能进行扩展

* 子类继承父类的语法格式如下：
* 修饰符 class subclass extends superclass {…..}
* 子类扩展了父类，将可以获得父类的属性和方法，但不能获得父类构造器
* Java里的继承都是单继承，也就是只能有一个直接的父类，可以有n个间接父类

创建子类对象的过程:递归创建父类对象，再创建子类对象,即先创建对应的顶级父类对象，然后依次创建相应的子类对象

* 通过关键字super 来调用父类的方法或属性

Super与this，他们的区别是？

* 子类构造器总会调用父类构造器。
* 如果子类构造器没有显式使用super调用父类构造器；子类构造器默认会调用父类无参数的构造器(如果父类没有无参数的构造器，子类必须显式调用父类有参构造器)

创建一个子类实例时，总会先调用最顶层父类的构造器

* 方法的重写要遵循“两同两小一大” 指的是：方法名相同，形参列表相同。返回值类型更小（继承关系，从父类到子类）或相同(object->string)，抛出的异常更小或相同，访问控制权限相同或更大 protected-》public
* @Override
* 方法重载：同一个类中，方法名相同，形参列表不同，与返回值无关

|  |
| --- |
| **public** **class** Animal {  **public** String name;  **public** String gender;  **public** **int** age;    **public** Animal(){  System.***out***.println("animal");  }  **public** Animal(String name, String gender, **int** age) {  System.***out***.println("animal arguments");  **this**.name = name;  **this**.gender = gender;  **this**.age = age;  }  // 如果一个类没有指明显示的继承关系，那么他继承自Object类  // Object是所有类的基/父类，Object对象，是java提供的一个类  // 我们不要在提供  @Override  **public** String toString() {  **return** "name=" + name + ", gender=" + gender + ", age=" + age ;  }  **protected** Object sleep(String time){  **return** "sleep" + time;  }  } |
| /\*\*  \* Bird子类  \* Animal是父类、超类、基类  \* Bird继承自Animal，那么Bird会继承Animal中的所有属性与方法  \* （private 可继承、不可以操作）  \* Bird直接继承自Animal，不能再直接继承其他类，继承是一种单继承的关系，  \* 任何类只能有一个直接父类  \* **@author** tzhang  \*/  **public** **class** Bird **extends** Animal {  // 子类区别于父类的属性  **public** String race;    **public** Bird(){  System.***out***.println("bird");  }  **public** Bird(String name,String gender,**int** age,String race) {  // 调用父类默认的构造器，不管写不写，都会调用  // 当写的时候，会调用所写的父类构造器，当不写时默认调用父类  // 空的构造器  **super**(name, gender, age);  System.***out***.println("brid arguments");  **this**.race = race;// 为什么不能放在super的上面？  }    **public** **void** fly(){  System.***out***.println(**this**.name + " is flying!---");  }  /\*\*  \* 重写父类的方法，意味着覆盖从父类继承过来的该方法  \* 有继承才有重写  \*/  @Override  **public** String toString() {  // super.toString()调用父类的toString方法  **return** "race=" + race + ", " + **super**.toString();  }  /\*\*  \* 重写方法声明都一样，没有问题  \* 方法的重写要遵循“两同两小一大” 指的是：方法名相同，形参列表相同。  \* 返回值类型更小（继承关系，从父类到子类）或相同，抛出的异常更小或相同，  \* 访问控制权限相同或更大  \*/  @Override  **public** String sleep(String time) {  System.***out***.println(name + " " + **super**.sleep(time));  **return** name + " " + **super**.sleep(time);  }  } |
| **public** **class** Main {  **public** **static** **void** main(String[] args) {  Animal animal = **new** Animal("Tiger", "tigeress",11);  System.***out***.println("=============");  System.***out***.println(animal.toString());  System.***out***.println("=============");    // 先创建Object-》Animal-》Bird  Bird bird = **new** Bird("sparrow","男",1,"little bird!");    System.***out***.println("=============");  System.***out***.println(bird.toString());    bird.fly();    System.***out***.println(bird.race);    Bird bird1 = **new** Bird();  bird1.sleep("8小時");    }  } |

多态Polymorphism

* Java 引用变量有两个类型：一个是编译时的类型，一个是运行时的类型，编译时的类型由声明该变量时使用的类型决定，运行时的类型由实际赋给该变量的对象决定。
* 如果编译时类型和运行时的类型不一致，这就有可能出现所谓的多态。
* 两个相同类型的引用变量，由于它们实际引用的对象的类型不同，当它们调用同名方法时，可能呈现出多种行为特征，这就是多态

将子类类型的引用赋值给父类类型的变量。

* 将一个子类对象赋给父类，这就是向上转型。向上转型是自动的
* 强制类型转换： 类型转换运算符是小括号，语法如下（type）variable，有父类型转换成子类型；
* 前一个操作通常是一个引用类型的变量，后一个操作通常是一个类（也可以是接口）。如果是返回true 否返回false。

组合（is-a）是在一类类中引用另一个类。生成另一个类的实例。

而继承（has-a）只是继承了父类的属性和方法。

区别：

使用组合可以用到另一个类中私有的属性和方法，而继承就不可以用到父类的私有的属性和方法了

他们都有各自的好处，要灵活的运用（根据实际场景）。

|  |
| --- |
| /\*\*  \* 车体大小，颜色，方向盘，轮胎  \* **@author** tzhang  \*  \*/  **public** **class** Car {  **public** String bodySize;  **public** String color;  **public** String steerwheel;  **public** String wheel;    **public** Car(){    }    **public** Car(String bodySize, String color, String steerwheel, String wheel) {  **super**();  **this**.bodySize = bodySize;  **this**.color = color;  **this**.steerwheel = steerwheel;  **this**.wheel = wheel;  }    **public** **void** userOil(String name) {  System.***out***.println(name + "燒油");  }  } |
| **public** **class** SamllCar **extends** Car{  **public** String smallTrunk;  **public** SamllCar() {  **super**();  // **TODO** Auto-generated constructor stub  }  **public** SamllCar(String bodySize, String color, String steerwheel, String wheel,String smallTrunk) {  **super**(bodySize, color, steerwheel, wheel);  // **TODO** Auto-generated constructor stub  **this**.smallTrunk = smallTrunk;  }  @Override  **public** **void** userOil(String name) {  System.***out***.println(name + "燒汽油");  }  **public** **void** booth(){  System.***out***.println("4个座");  }  } |
| **public** **class** Truck **extends** Car{  **public** String container;  **public** Truck() {  **super**();  // **TODO** Auto-generated constructor stub  }  **public** Truck(String bodySize, String color, String steerwheel, String wheel, String container) {  **super**(bodySize, color, steerwheel, wheel);  // **TODO** Auto-generated constructor stub  **this**.container = container;  }  @Override  **public** **void** userOil(String name) {  System.***out***.println(name + "燒柴油");  }  } |
| /\*\*  \* 两个相同类型的引用变量，由于它们实际引用的对象的类型不同，  \* 当它们调用同名方法时，可能呈现出多种行为特征，这就是多态  \* **@author** tzhang  \*  \*/  **public** **class** Main {  // 谈到多态，要说出这句话-》将子类类型的引用赋值给父类类型的变量  // 即父类变量指向具体的子类实例  **public** **static** **void** main(String[] args) {  // 编译时sc是Car类型， 编译时的类型由声明该变量时使用的类型决定  // 将子类赋值给父类，不需要转换  Car sc = **new** SamllCar("20平米", "blue",  "汽車方向盤", "汽車輪胎", "小後備箱");  // 運行時，sc是SamllCar類型,运行时的类型由实际赋给该变量的对象决定  sc.userOil("法拉利");    // 将父类赋值给子类时，需要强转类型转换  // SamllCar smallCar = (SamllCar) sc;  // smallCar.booth();    // 编译时truck是Car类型  Car truck = **new** Truck("40平米", "white",  "卡車方向盤", "卡車輪胎", "大貨箱");  // 運行時，truck是Truck類型  truck.userOil("東風");  // ClassCastException类型转换异常 SamllCar tr = (Truck) truck;  Truck tr = (Truck) truck;  // tr.booth();    // 判断前面的实例是否是后面类型的实例或者是后面类型子类的实例  System.***out***.println(sc **instanceof** SamllCar);// true  System.***out***.println(sc **instanceof** Car);// true  System.***out***.println(sc **instanceof** Truck); // false    **if**(sc **instanceof** SamllCar){  SamllCar tru = (SamllCar) sc;  tru.booth();  } **else** {  System.***out***.println("====");  }  }  } |