Java中的多态和类型转换

1. **Java中的多态**
2. 多态的进一步理解，多态可以理解为两个方面：

（1） 父类型的引用可以指向子类型的实例或对象，

（2） 接口类型的引用可以指向实现该接口的类的实例或对象（instance）。

**由此来看，以后一提到多态，应该马上映射到向上转型。**

1. 接口就是一种特殊的抽象类，关键字interface，实现用implements。例如：结合数组定义：

interface Interface1 implements Bb

{

}

Interface1[] = new Interface1[]{Bb(),Bb()};//数组

1. **对象的多态一般就是两种形态：父类形态与本类形态。**

**子类的对象就在这两种形态之间相互转换。**

**（1）向上转型是把子类对象转成父类形态，限制子类的特有方法；**

**（2）向下转型是把子类对象从父类形态转成本类形态（向下转型的前提是先进行向上转型）。**

1. Parent p = new Child（）；（1）当使用多态的方式调用方法时，首先检查父类中是否有此方法，如果没有，则会出现编译出错，如果有则可编译通过，包括子类和父类都有此方法，子类的把父类的覆盖，调用的是子类中的此方法，如果子类中不存在此方法，直接调用父类的方法。因此编译时要看父类中是否存在此方法，调用时具体要看所指向对象的中的方法，先看子类方法的重写，若没有再调用从父类继承下来的方法。此规律仅限于成员方法。（2）\*\*\*\*\*对于成员变量，则只看引用所属类中是否有此变量，调用的也是父类中的变量值，与子类中的变量没有一点儿关系。
2. 类类型的引用可以指向本类及其子类的对象；接口类型的引用可以用于指向实现接口的子类的对象。这就是多态的体现。
3. 这样让父类引用或接口引用作为形参，就可以接收很多子类的对象，拓展性比较强。
4. **多态：某一类事物的多种存在形态。**
5. 在java中多态的体现在：

（1） 父类的引用可以指向本类及其子类的对象；

（2）接口的引用可以指向实现此接口的类的对象。

1. 多态的好处：可以提高代码的复用性，提高代码的扩展性，前期定义的代码可以使用后期的内容。
2. 多态的弊端（局限性）：前期定义的内容不能使用（调用）后期子类的特有内容。
3. 使用多态的前提：**必须有关系：继承或者实现；要有覆盖（未经向下类型转换时，只能调用父类或者接口中有的，且被覆盖的方法）。**

**父类型的引用可以指向其子类的对象，子类有很多，在父类引用接收子类对象后，如何进行判断并向下转型呢？利用instanceof 进行判断，此关键词用于判断对象的具体类型，只可以用于引用数据类型的判断，如果父类引用指向子类对象，则父类引用instanceof 子类名如person Instanceof Student，返回true或者false。instanceof通常情况下在向下转型前用于健壮性的判断。**

**例如：public** **static** **void** specialMethod(Person p) {

**if**(p **instanceof** Student) {

Student s = (Student)p;

s.study();

}

**else** **if**(p **instanceof** Worker) {

((Worker)p).work();

}

}

1. **引入对象的多态后，对成员特点的分析：（十分重要）**

注意：重载和重写（覆盖）都是指的是方法的重载和重写，特别是当说覆盖时，指的也只是方法，成员变量没有覆盖这么一说，因为子类和父类中有重名的变量时，子类在实例化时会把这两个变量都进行封装，用特定的标号加以区分。

1. **成员变量：编译时，查看引用类型所属类中是否拥有需要访问的变量，若有则编译通过，否则编译不通过。**

**编译和执行（调用）都是看引用（左边）。**

**此时只关注引用类型所属类中是否有此变量，所指向的对象所属类有与没有此变量都无所谓。若都有，利用父类型引用调用的是父类型中的变量值。**

**对于成员变量，存在多态时，无论是编译还是执行调用，看的都是父类型引用所在的类，调用的变量也只是父类中的变量，父类中没有则编译不通过，子类中有与没有此变量都无关紧要。因为成员变量没有覆盖这个概念之说，覆盖指的只是方法。**

**实例：class** Animal{

**int** num = 6;//**如果没有这句，则编译出错**

}

**class** Dog **extends** Animal{

**int** num = 4;**//这句话有或者没有，输出都是6，因为多态时，对于成员变量，只关注父类的成员变量，子类的不关心。**

}

**public** **class** PolymorphismTest2 {

**public** **static** **void** main(String[] args) {

Animal animal = **new** Dog();

**int** num = animal.num;

System.***out***.println(num);

}

} 结果是：6

若在主函数中加上Dog dog = (Dog)animal;

System.***out***.println(dog.num); 若子类定义了num=4；则 此句话输出的就是4了。若子类中没有定义num变量，则输出的就是从父类继承下来的变量值，输出为6。

1. **成员方法：编译时，查看引用类型所属类中是否拥有需要访问的方法，若有则编译通过，否则编译不通过。编译看引用（类），执行（调用）看（所指）对象。**

**引用类中如果没有定义此方法，则编译不通过，调用执行时，首先调用子类中对此方法的重写方法，如没有重写，直接调用引用类中的此方法。（原因：非静态方法必须使用对象进行调用，静态方法可以不依赖对象）**

1. **静态方法：其实多态性指的是对象的多态性；但是静态方法根本不依赖于对象，所以严格意义上讲，静态方法不涉及对象的多态性。直接利用相应类名进行访问即可。对于静态方法，引用是什么类型，调用的就是什么类型的静态方法，与子类中的静态方法无关。**

**对于静态方法，编译和执行调用都是看引用类型。**

1. **分析：静态方法与成员变量一样，都是看引用类型。只有成员方法比较特殊，因为一般成员方法都是依赖于特定对象才能调用的。**
2. **Java中对象的转型**
3. **（必须理解透彻）注意:一个引用能够调用哪些成员（成员变量和成员方法），取决于这个引用本身是哪种类型，而调用的具体成员就是所指向对象的成员了（一个引用调用的是哪一个方法，取决于这个引用所指向的对象）。（通俗讲，就是能够调用哪些，看的是自身的类型，调用到底是什么，看的是指向的对象。）**
4. **上面的描述就把多态的弊端展露出来了：父类型的引用指向了子类的对象，但是不能调用子类特有的成员方法，限制对特有方法的调用，接口也如此。**
5. **对象的向上转型：**将子类的对象赋给父类的引用或者将父类的引用指向子类的对象；向上类型转换就限制对子类的特有方法的调用。
6. **对象的向下转型**：将父类的对象（需要强制类型转换）赋值给子类的引用或者将子类的引用指向父类的对象。只能是先进行向上转型，再进行向下转型才可以，不可以直接生成一个父类对象，对其直接向下转型（这是错误的）。

例如：正确的： Person p = new Student（）； //首先向上转型

Student s1 = (Student) p ; //向下转型

1. 注意：向下转移的前提是必须先向上转型，再向下转型。因为向下转型的引用必须是指向子类对象的，才可以进行向下转型。

错误的：Person p = new Person（）；

Student s1 = (Student) p ; //编译时可能不会出错，但是执行的时候是错误的。

1. 向上转型目的：提高扩展性，但是同时限制了子类特有功能的使用。

父类的引用指向子类的对象，但是此时能够调用哪些方法要看父类，具体调用什么方法看所指向的对象，因此子类中特有的方法就不能被调用。

向下转型的目的：为了使用子类中的特有方法。

1. 学生一定是人，但是人不一定是学生。
2. 对象转型分为向上转型和向下转型。上下如何区分的：父类为上，子类为下。
3. 向上转型的用处：由于父类的引用可以指向任何一个子类的对象，所以可以根据情况让同一个父类引用指向不同的子类的不同的对象。
4. 简单的说：转型都是对象的引用在进行指向的改变。
5. **向上转型**：记住：父类引用可以指向子类对象，不需要强制类型转换即可进行向上转型；

**向下转型**：记住：父类的引用必须是指向子类对象的，才可以进行向下转型为子类引用。

1. 向下类型转换：从父类可以转向所指向的子类类型。
2. **向上转型相当于把子类的特有属性隐藏掉了，若需要调用子类的特有属性，需要进行向下转型。**
3. 一共有两种类型的强制类型转换：

第一种是向上类型转换（upcast）：将子类型转换成父类型，不需要显示指定；如

Cat cat = new Cat（）；

Animal animal = cat；//向上类型转换，不需要显示指定

animal.sing;

第二种是向下类型转换（downcast）：将父类型转换成子类型，需要显示指定（必须使用强制类型转换）。如

Animal a = new Cat（）；

Cat cat = （Cat）a；

cat.sing;

1. 多态：综合：所谓多态就是，父类性的引用可以指向子类型的对象或者接口类型的引用可以指向实现该接口的类的实例或对象。关于接口与实现接口的类之间的强制类型转换方式与父类和子类之间的强制类型转换方式完全一样。
2. 如果类型转换出错，会抛出异常ClassCastException。
3. 对于转型，自始至终都是子类对象在做着类型的变化。

实例：**class** Person{

**public** **void** eat() {

System.***out***.println("person eat");

}

**public** **void** run() {

System.***out***.println("person run");

}

}

**class** Student **extends** Person{

**public** **void** study() {

System.***out***.println("student study");

}

**public** **void** eat() {

System.***out***.println("student eat");

}

}

**public** **class** TypeTrans {

**public** **static** **void** main(String[] args) {

Person p = **new** Student();//向上转型，但是不能利用p调用study方法

p.eat();//调用重写后的方法，具体调用哪个方法看指向的对象

p.run();//没有重写，直接调用从父类继承下来的run方法

Student s = (Student)p;//向下转型

s.study();

}

}