1、课程名称：对象的多态性

2、本课程预计讲解的知识点

本章目标

- 掌握对象的向上转型及向下转型的使用
- 掌握对象转型的限制

E-MAIL：mldnqa@163.com www.MLDNJAVA.cn

3、具体内容

3.1、对象多态性

多态性

- 多态性在面向对象中中是一个最重要的概念，在java中面向对象主要有以下两种主要体现：
  - 方法的重载与覆写
  - 对象的多态性

E-MAIL：mldnqa@163.com www.MLDNJAVA.cn

掌握对象多态性之后可以直接应用在抽象类和接口上。

对象的多态性

- 对象的多态性主要分为以下两种类型：
  - 向上转型：子类对象 → 父类对象
    - 对于向上转型，程序会自动完成，格式：
      - **对象向上转型：**父类 父类对象 = 子类实例 ;
  - 向下转型：父类对象 → 子类对象
    - 对于向下转型时，必须明确的指明要转型的子类类型，格式：
      - **对象向下转型：**子类 子类对象 = (子类)父类实例 ;

E-MAIL：mldnqa@163.com www.MLDNJAVA.cn

```
class A{                        // 定义类 A
    public void fun1(){         // 定义 fun1()方法
        System.out.println("A --> public void fun1(){}") ;
    }
    public void fun2(){
        this.fun1() ;           // 调用 fun1()方法
    }
};
class B extends A{
    public void fun1(){         // 此方法被子类覆写了
        System.out.println("B --> public void fun1(){}") ;
    }
    public void fun3(){
        System.out.println("B --> public void fun3(){}") ;
    }
```



```
};
public class PolDemo01{
    public static void main(String asrgs[]){
        B b = new B() ;         // 实例化子类对象
        A a = b ;               // 向上转型关系
        a.fun1() ;              // 此方法被子类覆写过
        a.fun2() ;
    }
};
```

对于以上的程序，是通过其子类进行父类对象的实例化操作的，则如果调用的方法被子类覆写过，则肯定调用被覆写过的方法。

注意点：

转型之后，因为操作的是父类对象，所以是无法找到在子类中定义的新方法。

将父类对象变为子类对象，称为向下转型，向下转型需要采用强制的手段

```
class A{                        // 定义类 A
    public void fun1(){         // 定义 fun1()方法
        System.out.println("A --> public void fun1(){}") ;
    }
    public void fun2(){
        this.fun1() ;           // 调用 fun1()方法
    }
};
class B extends A{
    public void fun1(){         // 此方法被子类覆写了
        System.out.println("B --> public void fun1(){}") ;
    }
    public void fun3(){
        System.out.println("B --> public void fun3(){}") ;
    }
};
public class PolDemo02{
    public static void main(String asrgs[]){
        A a = new B() ;             // 向上转型关系
        B b = (B)a ;            // 发生了向下转型关系
        b.fun1() ;
        b.fun2() ;
        b.fun3() ;
    }
```



```
};
```

在类 B 中存在三个方法，所以全部都可以调用。

但是，在进行对象向下转型操作的时候有一个注意点。

```
class A{                        // 定义类 A
    public void fun1(){         // 定义 fun1()方法
        System.out.println("A --> public void fun1(){}") ;
    }
    public void fun2(){
        this.fun1() ;           // 调用 fun1()方法
    }
};
class B extends A{
    public void fun1(){         // 此方法被子类覆写了
        System.out.println("B --> public void fun1(){}") ;
    }
    public void fun3(){
        System.out.println("B --> public void fun3(){}") ;
    }
};
public class PolDemo03{
    public static void main(String asrgs[]){
        A a = new A() ;             // 实例化了一个父类对象
        B b = (B)a ;            // 发生了向下转型关系
        b.fun1() ;
        b.fun2() ;
        b.fun3() ;
    }
};
```

发生的异常：

```
Exception in thread "main" java.lang.ClassCastException: A cannot be cast to B
        at PolDemo03.main(PolDemo03.java:20)
```

以上的异常是第二大出现的异常，此异常出现是在对象转型的时候经常发生的，如果两个没有关系的对象之间发生了转换关系，则肯定出现此异常。

也就是说，如果要想产生对象的向下转型，则肯定必须先产生一个向上的转型关系。“A a = new B() ;”表示建立关系。

3.2、对象多态性的应用

要求：设计一个方法，此方法可以接收 A 类的任意子类对象，并调用方法。

实现一：不使用对象多态性完成。使用重载完成

```
class A{                        // 定义类 A
    public void fun1(){         // 定义 fun1()方法
        System.out.println("A --> public void fun1(){}") ;
    }
    public void fun2(){
        this.fun1() ;           // 调用 fun1()方法
    }
};
class B extends A{
    public void fun1(){         // 此方法被子类覆写了
        System.out.println("B --> public void fun1(){}") ;
    }
    public void fun3(){
        System.out.println("B --> public void fun3(){}") ;
    }
};
class C extends A{
    public void fun1(){         // 此方法被子类覆写了
        System.out.println("C --> public void fun1(){}") ;
    }
    public void fun5(){
        System.out.println("C --> public void fun5(){}") ;
    }
};
public class PolDemo04{
    public static void main(String asrgs[]){
        fun(new B()) ;   // 传递 B 的实例
        fun(new C()) ;   // 传递 B 的实例
    }
    public static void fun(B b){
        b.fun1() ;          // 调用覆写父类中的 fun1()方法
    }
    public static void fun(C c){
```



```
        c.fun1() ;          // 调用覆写父类中的 fun1()方法
    }
};
```

如果按照以上的设计思路，每增加一个子类，则 fun()方法就必须重载一次，如果现在 A 类有 10000 个子类呢？

此时，为了解决这样的问题，就可以使用对象的多态性完成操作。

```
class A{                        // 定义类 A
    public void fun1(){         // 定义 fun1()方法
        System.out.println("A --> public void fun1(){}") ;
    }
    public void fun2(){
        this.fun1() ;           // 调用 fun1()方法
    }
};
class B extends A{
    public void fun1(){         // 此方法被子类覆写了
        System.out.println("B --> public void fun1(){}") ;
    }
    public void fun3(){
        System.out.println("B --> public void fun3(){}") ;
    }
};
class C extends A{
    public void fun1(){         // 此方法被子类覆写了
        System.out.println("C --> public void fun1(){}") ;
    }
    public void fun5(){
        System.out.println("C --> public void fun5(){}") ;
    }
};
public class PolDemo05{
    public static void main(String asrgs[]){
        fun(new B()) ;   // 传递 B 的实例
        fun(new C()) ;   // 传递 B 的实例
    }
    public static void fun(A a){
        a.fun1() ;          // 调用覆写父类中的 fun1()方法
    }
```



```
};
```

如果采用以上的形式，不管有多少个子类，代码都可以轻松的完成。

4、总结

1、 对象多态性的概念
   - 向上转型：自动完成。
   - 向下转型：强制。发生向下转型关系之前必须先发生向上的转型关系
2、 对象多态性可以解决方法接收参数的问题。

5、预习任务

下一章内容

E-MAIL：mldnqa@163.com www.MLDNJAVA.cn