多态（Polymorphism）

多态概述

> Java有三大特性：封装、继承和多态。

那么什么是多态呢？所谓多态就是指程序中定义的引用变量所指向的具体类型和通过该引用变量发出的方法调用在编程时并不确定，而是在程序运行期间才确定，即一个引用变量倒底会指向哪个类的实例对象，该引用变量发出的方法调用到底是哪个类中实现的方法，必须在由程序运行期间才能决定。

因为在程序运行时才确定具体的类，这样，不用修改源程序代码，就可以让引用变量绑定到各种不同的类实现上，从而导致该引用调用的具体方法随之改变，即不修改程序代码就可以改变程序运行时所绑定的具体代码，让程序可以选择多个运行状态，这就是多态性。

简单来说就是根据不同情况调用不同的方法。

背景知识

在了解多态之前，让我们先了解一下类型转换，因为多态本身是一种向上转型。

强制类型转换（cast）

```
double Steve_speed = 233.33;
int now_Steve_speed = (int) Steve_speed;
```

向上转型（upcasting）and 向下转型（downcasting）

```
public class Minecraft {
    public void sleep() {
        System.out.println("Set time night.");
    }

    public static void main(String[] args) {
        Steve m = new Steve();
        m.sleep();

        Minecraft h = new Steve();// （1）向上转型
        h.sleep(); // （2）动态绑定
        // h.speak(); // （3）此方法不能编译，报错说Minecraft类没有此方法
```

```
        //          // [1]向下转型
        //          Steve m2 = new Steve();
        //          Minecraft h2 = m2;
        //          m2 = (Steve) h2;
        //          m2.speak();
        //
        //          // [2]向下转型：失败
        //          Minecraft h3 = new Minecraft();
        //          Steve m3 = (Steve)h3;
        //          m3.speak();     //此时会出现运行时错误，所以可以用instanceOF判断
        //
        //          // [3]向下转型：类型防护
        //          Minecraft h4 = new Minecraft();
        //          if (h4 instanceof Steve){ // 因为h4不是Steve的实例，所以不执行if内部代码
        //              Steve m4 = (Steve)h4;
        //              m4.speak();
        //          }

    }
}

class Steve extends Minecraft {
    @Override
    public void sleep() {
        System.out.println("Steve sleep.");
    }

    public void speak() {
        System.out.println("I am Steve.");
    }
}

class Alex extends Minecraft {
    @Override
    public void sleep() {
        System.out.println("Alex sleep.");
    }

    public void speak() {
        System.out.println("I am Alex.");
    }
}
```

- 这里要注意：

1. 向上转型不要强制转型
2. 父类引用指向的或者调用的方法是子类的方法,这个叫动态绑定
3. 向上转型后父类引用不能调用子类自己的方法

知识点详述（实例）

> 多态性（Polymorphism）：向上转型

在搜索资料的过程中，发现有人说： 重载也是一种多态，不过是一种特殊的多态，是编译时决定的静态多态。但网上又有人说重载是多态的静态绑定，重写是多态的动态绑定。

其实java里方法被封装后也可以是对象，多态的必要条件虽然可以理解是继承和重写，但多态分为行为(方法)多态和对象多态，行为多态是重写和重载,对象多态是向上下转型。 简单的说，重写是父类与子类之间多态性的体现，而重载是一个类的行为的多态性的体现。

而我们平常讲的java的三大特性之多态多指多态的动态绑定，有三个必要条件：

- 继承
- 重写
- 父类引用指向子类对象（向上转型）

```
public class Minecraft {
    public static void main(String[] args) {
        Mineral p = new Iron();
        p.mine();
        // 调用特有的方法
        Iron s = (Iron) p;
        s.broked();
        // ((Iron) p).broked();
    }
}

class Mineral {
    public void mine() {
        System.out.println("矿物");
    }
}

class Iron extends Mineral {
    @Override
    public void mine() {
        System.out.println("铁矿");
    }

    public void broked() {
        System.out.println("石镐可以破坏它");
    }
}

class Diamond extends Mineral {
    @Override
    public void mine() {
        System.out.println("钻石");
    }

    public void what() {
        System.out.println("掉落钻石");
    }
}
```

- 输出：
- ```
  铁矿
  石镐可以破坏它
  ```

多态-使用场景

> 多态：基类型对象访问派生类重写的方法
  ```

访问方式

- 循环调用基类对象，访问不同派生类方法
- 自定义函数，实参是派生类，行参是基类

多态的优点

- 消除类型之间的耦合关系
- 可替换性
- 可扩充性
- 接口性
- 灵活性
- 简化性

小结

多态虽然有点难以理解，但其特性动态绑定在实际项目中非常有用，能减少很多繁杂的代码，减轻工作量的同时，其灵活性也大大提高了代码的扩展性。