面向对象

封装

含义：

指的是将数据（变量）和操作数据的方法绑定在一起，并隐藏对象的内部状态，使其对外部不可见或受控访问。通过这种方式，封装提高了代码的安全性、可维护性和灵活性。

目的:

- 数据隐藏：封装允许隐藏类的内部实现细节，只暴露必要的部分给外部使用。这通常通过将类成员（属性和方法）设置为私有（`private`），并提供公共（`public`）的getter和setter方法来访问和修改这些成员。
- 控制访问级别：通过访问修饰符（public，protected,默认,private）保证只有对应的范围内能够访问和修改某些数据,从而保证了安全性。
- 增加灵活性：当类的内部实现发生改变时，如果对外接口保持不变，则不需要更改使用该类的所有地方。例如，如果你决定更改某个属性的存储方式或计算方式，只需在对应的getter和setter方法中做出调整即可，而不需要改动所有调用该属性的地方。
- 增加一些限制：比如在以下代码中的setAge（）方法中对年龄进行筛选

举例：

```
public class Person {
    // 私有字段，外部无法直接访问
    private String name;
    private int age;

    // 公共的getter和setter方法
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        if (age > 0) { // 添加一些基本的验证
            this.age = age;
        } else {
            System.out.println("年龄必须大于0");
        }
    }
}
```

继承

含义：

当类与类之间，存在相同（共性）的内容，并满足子类是父类中的一种，就可以考虑继承。

目的

- 可以把多个子类中重复的代码提取到父类中，提高代码的复用性
- 可以在父类的基础上，在子类中增加其他功能

特点：

- java中只支持单继承，不支持多继承，但支持多层继承。
  - 单继承：一个子类只能继承一个父类
  - 不支持多继承：子类不能同时继承多个父类。
  - 多层继承：子类A继承父类B，父类B可以继承父类C。而父类B是子类A的直接父类，父类C是子类A的间接父类。
- java中所有的类都直接或间接继承于object类。

重写:

- 只有虚方法(非private，非static，非final)才能被重写
- 重写的方法名称和形参列表必须与父类一致
- 子类重写时，访问权限子类必须大于等于父类；返回值类型必须小于等于父类

####

多态:

含义:

同类型的对象，表现出的不同形态

```
//父类类型 对象名称=子类对象
FU f=new Zi();
```

多态调用成员的特点：

- 变量调用：编译看左边，运行看左边
- 方法调用：编译看左边，运行看右边

```java
public static void main(String[] args) {
    //创建对象（多态方式）
    //Fu f = new Zi();
    Animal a = new Dog();
    //调用成员变量：编译看左边，运行也看左边
    //编译看左边：javac编译代码的时候，会看左边的父类中有没有这个变量，如果有，编译成功，如果没有编译失败。
    //运行也看左边：java运行代码的时候，实际获取的就是左边父类中成员变量的值
    System.out.println(a.name);//动物

    //调用成员方法：编译看左边，运行看右边
    //编译看左边：javac编译代码的时候，会看左边的父类中有没有这个方法，如果有，编译成功，如果没有编译失败。
    //运行看右边：java运行代码的时候，实际上运行的是子类中的方法。
    a.show();///Dog --- show方法
}
```

多态的优势和弊端：

- 优势：方法中，使用父类型作为参数，可以接收所有子类对象
- 弊端：由于编译看左边，运行看右边，所以虚拟机在左边（父类）中搜索方法，如果这个方法仅仅是子类有而父类没有，那么虚拟机直接报错。
- 解决方案:将变量变为子类型，使用**强制转换**

```java
//先判断a是否为Dog类型，如果是，则强转成Dog类型，转换之后变量名为d
//如果不是，则不强转，结果返回false
if(a instanceof Dog d){
    d.方法;
}else if(a instanceof Cat c){
    c.方法;
}else{
    sout("没有这个类型，无法转换");
}
```

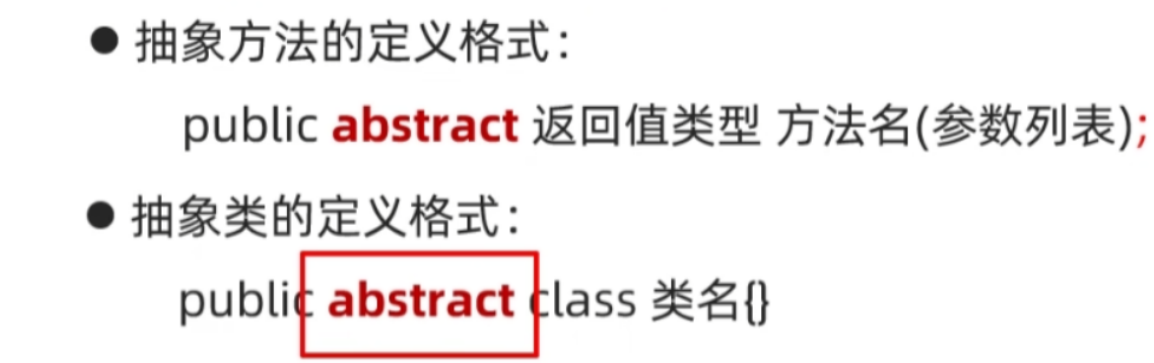
抽象：

抽象方法

- 抽象方法：将**共性的**行为（**方法**）抽取到父类之后。
  由于每一个子类执行的内容是不一样，
  所以，在父类中不能确定**具体的方法体**。
  该方法就可以定义为抽象方法。

- 抽象类：如果一个**类中存在抽象方法**，那么该类就**必须声明为抽象类**

抽象类和抽象方法的定义格式



注意事项:

- 抽象类不能实例化（创建对象）
- 抽象类中不一定有抽象方法，有抽象方法一定是抽象类
- 可以有构造方法（虽然抽象类不能创建对象，但是当创建子类对象时，可以给属性进行赋值）
- 抽象类的子类：（一般使用第一种）
  - 要么重写抽象类中的所有抽象方法
  - 要么是抽象类