## 七大设计原则

# 单一职责原则

# 接口隔离原则

# 依赖倒转原则

# 里式替换原则

# 开闭原则

## 定义：对扩展开放，对修改关闭

## 示例

### 反例：

|  |
| --- |
| */\*\*  \* 功能描述：  \*  \** ***@Author*** *Tracy  \** ***@Date*** *2020/7/1 14:32  \*/* **public class** Ocp {  **public static void** main(String[] args) {  GraphicEditor graphicEditor = **new** GraphicEditor();  graphicEditor.drawShape(**new** Rectangle());  graphicEditor.drawShape(**new** Circle());  graphicEditor.drawTriangle(**new** Triangle());  } } *//这是一个用于绘图的类* **class** GraphicEditor{  *//接收Shape对象，然后根据type,来绘制不同的图形* **public void** drawShape(Shape s){  **if**(s.**m\_type** == 1){  drawRectangle(s);  }  **else if**(s.**m\_type** == 2){  drawCircle(s);  }  **else if**(s.**m\_type** == 3){  drawTriangle(s);  }  }  *//绘制矩形* **public void** drawRectangle(Shape r){  System.***out***.println(**"绘制矩形"**);  }  *//绘制圆形* **public void** drawCircle(Shape r){  System.***out***.println(**"绘制圆形"**);  }  *//绘制三角形* **public void** drawTriangle(Shape r){  System.***out***.println(**"绘制三角形"**);  } }  **class** Shape{  **int m\_type**; }  **class** Rectangle **extends** Shape{  **public** Rectangle() {  **m\_type** = 1;  } }  **class** Circle **extends** Shape{  **public** Circle() {  **m\_type** = 2;  } }  **class** Triangle **extends** Shape{  **public** Triangle(){  **m\_type** = 3;  } } |

### 正例：

|  |
| --- |
| **package** com.tracy.principle.ocp.improve;  */\*\*  \* 功能描述：  \*  \** ***@Author*** *Tracy  \** ***@Date*** *2020/7/1 14:32  \*/* **public class** Ocp {  **public static void** main(String[] args) {  GraphicEditor graphicEditor = **new** GraphicEditor();  graphicEditor.drawShape(**new** Rectangle());  graphicEditor.drawShape(**new** Circle());  graphicEditor.drawShape(**new** Triangle());  graphicEditor.drawShape(**new** OtherGraphic());  } }  *//这是一个用于绘图的类* **class** GraphicEditor{  *//接收Shape对象，调用draw方法* **public void** drawShape(Shape s){  s.draw();  } }  **abstract class** Shape{  **int m\_type**;  **public abstract void** draw(); }  *//画矩形* **class** Rectangle **extends** Shape {  **public** Rectangle() {  **m\_type** = 1;  }   @Override  **public void** draw() {  System.***out***.println(**"绘制矩形"**);  } }  *//画圆形* **class** Circle **extends** Shape {  **public** Circle() {  **m\_type** = 2;  }   @Override  **public void** draw() {  System.***out***.println(**"绘制圆形"**);  } }  *//新增画三角形* **class** Triangle **extends** Shape {  **public** Triangle(){  **m\_type** = 3;  }   @Override  **public void** draw() {  System.***out***.println(**"绘制三角形"**);  } } *//新增一个图形* **class** OtherGraphic **extends** Shape{  **public** OtherGraphic() {  **m\_type** = 4;  }   @Override  **public void** draw() {  System.***out***.println(**"绘制其他图形"**);  } } |

## 总结：

图像提供方预定图形 》》 图像提供方扩展图形 》》 图形使用方绘制图形

图像提供方通过编号 m\_type来扩展不同的图形

图像使用方通过编号m\_type来选择绘制哪种图形，即调用哪个图形的绘制方法，扩展绘制不同图形的方法

改进：

图像提供方预定义图形及图形绘制方法 （抽象父类和抽象方法） >>

图像提供方扩展图形 则通过继承抽象父类，重写抽象方法 来定义图形编号m\_type和绘制方法

图像使用方 通过依赖抽象类来实现图像绘制

# 迪米特法则（最少知道原则）

## 定义：对自己依赖的类，知道的越少越好

## 示例:

### 反例：

|  |
| --- |
| **package** com.tracy.principle.demeter;  **import** java.util.ArrayList; **import** java.util.List;  */\*\*  \* 功能描述：  \*  \** ***@Author*** *Tracy  \** ***@Date*** *2020/7/2 13:08  \*/  //client* **public class** Demeter {  **public static void** main(String[] args) {  *//创建了一个SchoolManager对象* SchoolManager schoolManager = **new** SchoolManager();  *//输出学院的员工Id和学校总部的员工信息* schoolManager.printAllEmployee(**new** CollegeManager());  } }  *//学校总部员工类* **class** Employee{  **private** String **id**;   **public void** setId(String id){  **this**.**id** = id;  }   **public** String getId() {  **return id**;  } } *//学院员工类* **class** CollegeEmployee{  **private** String **id**;   **public** String getId() {  **return id**;  }   **public void** setId(String id) {  **this**.**id** = id;  } } *//管理学院员工的管理类* **class** CollegeManager{  *//返回学院的所有员工* **public** List<CollegeEmployee> getAllEmployee(){  List<CollegeEmployee> list = **new** ArrayList<>();  **for** (**int** i = 0; i < 10; i++) {  CollegeEmployee emp = **new** CollegeEmployee();  emp.setId(**"学院员工Id = "** +i);  list.add(emp);  }  **return** list;  } } *//学校的管理类 //分析SchoolManager类的直接朋友类有哪些 Employee,CollegeManager(成员变量，方法参数，返回值) //CollegeEmployee 不是直接朋友 而是一个陌生类，这样违背了迪米特法则* **class** SchoolManager{  *//返回学校总部的员工* **public** List<Employee> getAllEmployee(){  List<Employee> list = **new** ArrayList<>();  **for** (**int** i = 0; i < 5 ;i++) {  Employee emp = **new** Employee();  emp.setId(**"学院总部员工Id = "** +i);  list.add(emp);  }  **return** list;  }  *//该方法完成输出学校总部和学院总部的员工* **void** printAllEmployee(CollegeManager sub){  *//分析问题  //1.这里的CollegeEmployee不是SchoolManager的直接朋友  //2.CollegeEmployee 是以局部变量方式出现在SchoolManager  //3.违反了迪米特法则   //分析问题  //1.将输出学院的员工方法，封装到CollegeManager  //获取到学院的员工* List<CollegeEmployee> list1 = sub.getAllEmployee();  System.***out***.println(**"---------学院员工----------"**);  **for** (CollegeEmployee e:list1){  System.***out***.println(e.getId());  }  List<Employee> list2 = **this**.getAllEmployee();  System.***out***.println(**"---------学校总部员工----------"**);  **for** (Employee e:list2){  System.***out***.println(e.getId());  }  } } |

### 正例：

|  |
| --- |
| **package** com.tracy.principle.demeter;  **import** java.util.ArrayList; **import** java.util.List;  */\*\*  \* 功能描述：  \*  \** ***@Author*** *Tracy  \** ***@Date*** *2020/7/2 13:08  \*/  //client* **public class** Demeter {  **public static void** main(String[] args) {  System.***out***.println(**"使用迪米特法则的改进"**);  *//创建了一个SchoolManager对象* SchoolManager schoolManager = **new** SchoolManager();  *//输出学院的员工Id和学校总部的员工信息* schoolManager.printAllEmployee(**new** CollegeManager());  } }  *//学校总部员工类* **class** Employee{  **private** String **id**;   **public void** setId(String id){  **this**.**id** = id;  }   **public** String getId() {  **return id**;  } } *//学院员工类* **class** CollegeEmployee{  **private** String **id**;   **public** String getId() {  **return id**;  }   **public void** setId(String id) {  **this**.**id** = id;  } } *//管理学院员工的管理类* **class** CollegeManager{  *//返回学院的所有员工* **public** List<CollegeEmployee> getAllEmployee(){  List<CollegeEmployee> list = **new** ArrayList<>();  **for** (**int** i = 0; i < 10; i++) {  CollegeEmployee emp = **new** CollegeEmployee();  emp.setId(**"学院员工Id = "** +i);  list.add(emp);  }  **return** list;  }  **public void** printEmployee(){  List<CollegeEmployee> list1 = getAllEmployee();  System.***out***.println(**"---------学院员工----------"**);  **for** (CollegeEmployee e:list1){  System.***out***.println(e.getId());  }  }  } *//学校的管理类 //分析SchoolManager类的直接朋友类有哪些 Employee,CollegeManager(成员变量，方法参数，返回值) //CollegeEmployee 不是直接朋友 而是一个陌生类，这样违背了迪米特法则* **class** SchoolManager{  *//返回学校总部的员工* **public** List<Employee> getAllEmployee(){  List<Employee> list = **new** ArrayList<>();  **for** (**int** i = 0; i < 5 ;i++) {  Employee emp = **new** Employee();  emp.setId(**"学院总部员工Id = "** +i);  list.add(emp);  }  **return** list;  }  *//该方法完成输出学校总部和学院总部的员工* **void** printAllEmployee(CollegeManager sub){  *//分析问题  //1.这里的CollegeEmployee不是SchoolManager的直接朋友  //2.CollegeEmployee 是以局部变量方式出现在SchoolManager  //3.违反了迪米特法则   //分析问题  //1.将输出学院的员工方法，封装到CollegeManager  //获取到学院的员工  /\* List<CollegeEmployee> list1 = sub.getAllEmployee();  System.out.println("---------学院员工----------");  for (CollegeEmployee e:list1){  System.out.println(e.getId());  }\*/* sub.printEmployee();  List<Employee> list2 = **this**.getAllEmployee();  System.***out***.println(**"---------学校总部员工----------"**);  **for** (Employee e:list2){  System.***out***.println(e.getId());  }  } } |

## 总结：

# 合成复用原则

尽量使用合成/聚合的方式，而不要使用继承的方式