# 简单工厂模式

|  |
| --- |
| **public** **class** OperationFactory {  **public** **static** Operation createOperate(String operate){  Operation oper = **null**;  **switch**(operate){  **case** "+":  oper = **new** OperationAdd();  **break**;  **case** "-":  oper = **new** OperationSub();  **break**;  **case** "\*":  oper = **new** OperationMul();  **break**;  **case** "/":  oper = **new** OperationDiv();  **break**;  }  **return** oper;  }  } |
| Operation oper = OperationFactory.*createOperate*("+");  oper.NumberA = 1;  oper.NumberB = 2;  **double** result = oper.GetResult(); |

# 2. 策略模式

# 3. 单一职责原则

# 4. 开放-封闭原则

# 5. 以来倒转原则

# 6. 装饰模式

# 7. 代理模式

# 8. 工厂方法模式

# 9. 原型模式

# 10. 模板方法模式

# 11. 迪米特法则

# 12. 外观模式

# 13. 建造者模式

# 14. 观察者模式

# 15. 抽象工厂模式

# 16. 状态模式

# 17. 适配器模式

# 18. 备忘录模式

# 19. 组合模式

# 20. 迭代器模式

# 21. 单例模式

# 22. 桥接模式

# 23. 命令模式

# 24. 职责链模式

# 25. 中介者模式

# 26. 享元模式

# 27. 解释器模式

# 28. 访问者模式

# ------------未分类-----------

|  |
| --- |
| 可维护(好修改)  可复用(重复用)  可扩展(方便添加东西)  灵活性好(方便用户自定义) |
| 通过封装、继承、多态. 降低程序的耦合度 |