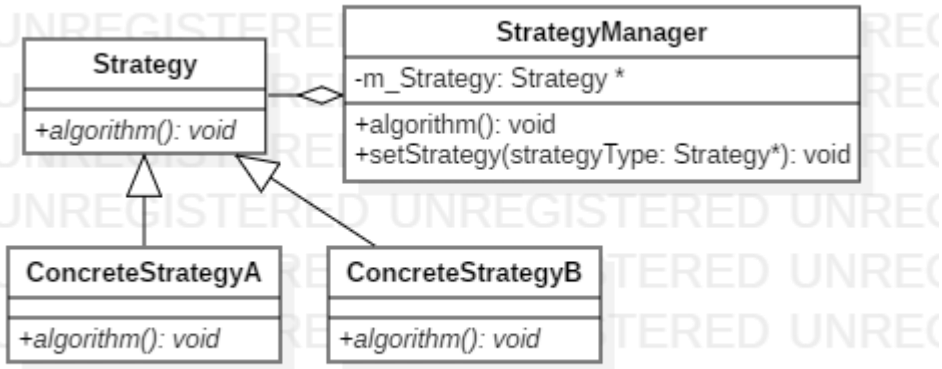
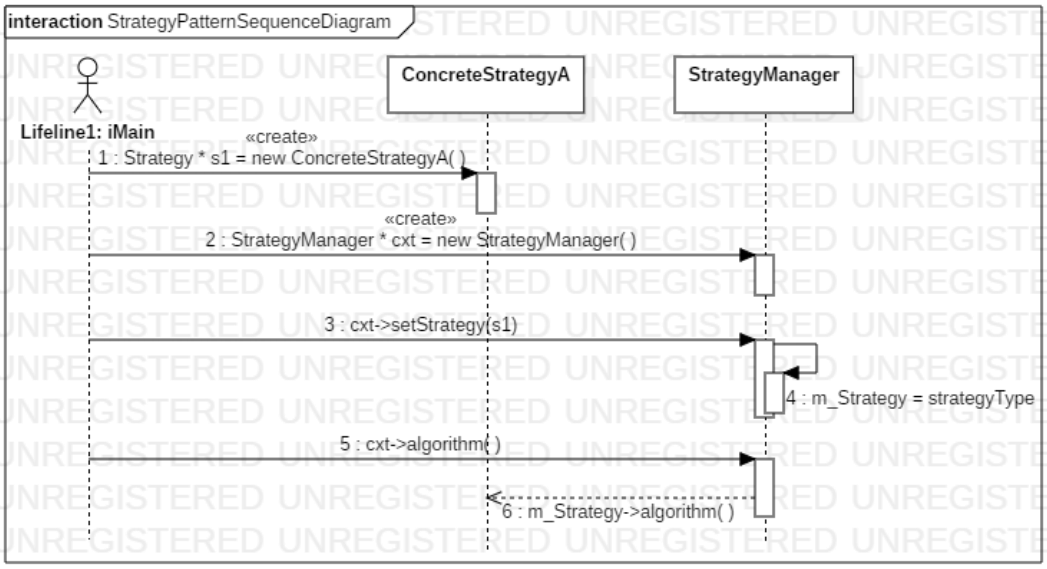
模式定义

1. 定义一个抽象算法类，被一系列算法，封装每个算法，同时这些算法可以互换，使得这些算法独立于用户，最终达到随用户而变化。
2. 继承抽象类 Strategy 的算法类 ConcreteStrategyA 和 ConcreteStrategyB，可以通过管理算法类 StrategyManager 进行替换算法

模式结构



时序图



关键代码

```
int main(int argc, char *argv[])
{
    Strategy * s1 = new ConcreteStrategyA();
    StrategyManager * cxt = new StrategyManager();
    cxt->setStrategy(s1);
    cxt->algorithm();

    Strategy *s2 = new ConcreteStrategyB();
    cxt->setStrategy(s2);
    cxt->algorithm();

    delete s1;
    delete s2;

    system("pause");
    return 0;
}
```

```
void StrategyManager::setStrategy(Strategy* strategyType)
{
    m_Strategy = strategyType;
}
```

```
void StrategyManager::algorithm()
{
    m_Strategy->algorithm();
}
```

```
void ConcreteStrategyA::algorithm()
{
    std::cout << "ConcreteStrategyA algorithm " << std::endl;
}
```

测试结果

```
int main(int argc, char *argv[])
{
    Strategy * s1 = new ConcreteStrategyA();
    StrategyManager * cxt = new StrategyManager();
    cxt->setStrategy(s1);
    cxt->algorithm();

    Strategy *s2 = new ConcreteStrategyB();
    cxt->setStrategy(s2);
    cxt->algorithm();

    delete s1;
    delete s2;

    system("pause");
    return 0;
```

```
D:\Design Pattern1\DesignPatterDem
ConcreteStrategyA algorithm
ConcreteStrategyB algorithm
请按任意键继续. . .
```