In [4]:

```cpp
#include <iostream>

// AbstractClass outlines the structure of an algorithm
class AbstractClass {
public:
    // The template method defines the sequence of steps to perform the algorithm.
    void templateMethod() const {
        baseOperation1();
        requiredOperations1();
        baseOperation2();
        hook1();
        requiredOperations2();
        baseOperation3();
        hook2();
    }

protected:
    // Common implementations provided by the AbstractClass
    void baseOperation1() const {
        std::cout << "AbstractClass says: I am doing the bulk of the work\n";
    }
    void baseOperation2() const {
        std::cout << "AbstractClass says: But I let subclasses override some operat
    }
    void baseOperation3() const {
        std::cout << "AbstractClass says: But I am doing the bulk of the work anywa
    }

    // Abstract operations that must be implemented by subclasses
    virtual void requiredOperations1() const = 0;
    virtual void requiredOperations2() const = 0;

    // Hooks provide additional extension points
    virtual void hook1() const {}
    virtual void hook2() const {}
};

// ConcreteClass1 and ConcreteClass2 implement the algorithm steps differently
class ConcreteClass1 : public AbstractClass {
protected:
    void requiredOperations1() const override {
        std::cout << "ConcreteClass1 says: Implemented Operation1\n";
    }
    void requiredOperations2() const override {
        std::cout << "ConcreteClass1 says: Implemented Operation2\n";
    }
};

class ConcreteClass2 : public AbstractClass {
protected:
    void requiredOperations1() const override {
        std::cout << "ConcreteClass2 says: Implemented Operation1\n";
    }
    void requiredOperations2() const override {
```

```
        std::cout << "ConcreteClass2 says: Implemented Operation2\n";
    }
    void hook1() const override {
        std::cout << "ConcreteClass2 says: Overridden Hook1\n";
    }
};

void clientCode(AbstractClass* class_) {
    class_->templateMethod();
}

int main() {
    std::cout << "Demonstrating Template Method with different subclasses:\n";
    ConcreteClass1* concreteClass1 = new ConcreteClass1;
    clientCode(concreteClass1);
    std::cout << "\n";
    ConcreteClass2* concreteClass2 = new ConcreteClass2;
    clientCode(concreteClass2);

    delete concreteClass1;
    delete concreteClass2;
    return 0;
}
```

```
Demonstrating Template Method with different subclasses:
AbstractClass says: I am doing the bulk of the work
ConcreteClass1 says: Implemented Operation1
AbstractClass says: But I let subclasses override some operations
ConcreteClass1 says: Implemented Operation2
AbstractClass says: But I am doing the bulk of the work anyway

AbstractClass says: I am doing the bulk of the work
ConcreteClass2 says: Implemented Operation1
AbstractClass says: But I let subclasses override some operations
ConcreteClass2 says: Overridden Hook1
ConcreteClass2 says: Implemented Operation2
AbstractClass says: But I am doing the bulk of the work anyway
```