DIP (Dependency Inversion Principle)  
  
// DIP\_Example\_CPP

#include <iostream>

#include <string>

//

// Dependency Inversion Principle with Dependency Injection

//

class IMessageService {

public:

virtual void sendMessage(const std::string& message) = 0;

virtual ~IMessageService() = default;

};

class EmailService : public IMessageService {

public:

void sendMessage(const std::string& message) override {

std::cout << "[EmailService] Sending Email: " << message << std::endl;

}

};

class SMSService : public IMessageService {

public:

void sendMessage(const std::string& message) override {

std::cout << "[SMSService] Sending SMS: " << message << std::endl;

}

};

class Notification {

private:

IMessageService\* service;

public:

Notification(IMessageService\* svc) : service(svc) {}

void notifyUser() {

service->sendMessage("Hello from DIP!");

}

};

//

// Factory Pattern

//

class ServiceFactory {

public:

static IMessageService\* createService(const std::string& type) {

if (type == "email") return new EmailService();

else if (type == "sms") return new SMSService();

else return nullptr;

}

};

//

// Strategy Pattern

//

class Strategy {

public:

virtual void execute() = 0;

virtual ~Strategy() = default;

};

class QuickSort : public Strategy {

public:

void execute() override {

std::cout << "[Strategy] Sorting using QuickSort..." << std::endl;

}

};

class BubbleSort : public Strategy {

public:

void execute() override {

std::cout << "[Strategy] Sorting using BubbleSort..." << std::endl;

}

};

class SortContext {

private:

Strategy\* strategy;

public:

void setStrategy(Strategy\* s) {

strategy = s;

}

void sort() {

if (strategy) strategy->execute();

else std::cout << "No sorting strategy set." << std::endl;

}

};

//

// Main

//

int main() {

std::cout << "=== Dependency Injection Example ===" << std::endl;

EmailService email;

Notification notif(&email);

notif.notifyUser();

std::cout << "\n=== Factory Pattern Example ===" << std::endl;

IMessageService\* service = ServiceFactory::createService("sms");

Notification notif2(service);

notif2.notifyUser();

delete service;

std::cout << "\n=== Strategy Pattern Example ===" << std::endl;

SortContext context;

QuickSort quick;

BubbleSort bubble;

context.setStrategy(&quick);

context.sort();

context.setStrategy(&bubble);

context.sort();

return 0;

}