**Practice Problems**

**Apply SOLID principles on following pieces of code:**

**Q1)**

**class Employee {**

**private:**

**string name;**

**double salary;**

**public:**

**void calculateTax();**

**void generateReport();**

**};**

**Solution:**

**Single Responsibility Principle (SRP):**

Refactor the Employee class to adhere to SRP by separating concerns:

class Employee {

private:

string name;

double salary;

public:

void calculateTax();

};

class ReportGenerator {

public:

void generateReport(Employee employee);

};

**Q2)**

class PaymentProcessor {

public:

void processPayment(Payment\* payment) {

if (payment->type() == "CreditCard") {

// Process credit card payment.

} else if (payment->type() == "PayPal") {

// Process PayPal payment.

} else {

// Unknown payment type.

}

}

};

**Solution:**

**Open Closed Principle:**

class Payment {

public:

virtual ~Payment() {}

virtual void process() = 0;

};

class CreditCardPayment : public Payment {

public:

void process() override {

// Process credit card payment.

}

};

class PayPalPayment : public Payment {

public:

void process() override {

// Process PayPal payment.

}

};

class BitcoinPayment : public Payment {

public:

void process() override {

// Process Bitcoin payment.

}

};

class PaymentProcessor {

public:

void processPayment(Payment\* payment) {

payment->process();

}

};

**Q3)**

**class Machine {**

**public:**

**virtual void print() = 0;**

**virtual void fax() = 0;**

**};**

**class Printer : public Machine {**

**public:**

**void print() override {**

**// Print implementation**

**}**

**void fax() override {**

**// Fax implementation**

**}**

**};**

**class FaxMachine : public Machine {**

**public:**

**void print() override {**

**// Print implementation**

**}**

**void fax() override {**

**// Fax implementation**

**}**

**};**

**Solution:**

**Interface Segregation Principle:**

class Printable {

public:

virtual void print() = 0;

};

class Faxable {

public:

virtual void fax() = 0;

};

class Printer : public Printable {

public:

void print() override {

// Print implementation

}

};

class FaxMachine : public Faxable {

public:

void fax() override {

// Fax implementation

}

};

**Q4)**

**class LightBulb {**

**public:**

**void turnOn() {**

**std::cout << "Light bulb is on." << std::endl;**

**}**

**void turnOff() {**

**std::cout << "Light bulb is off." << std::endl;**

**}**

**};**

**class LightSwitch {**

**public:**

**void operate() {**

**LightBulb bulb;**

**bulb.turnOn();**

**// ... additional logic ...**

**bulb.turnOff();**

**}**

**};**

**int main() {**

**LightSwitch lightSwitch;**

**lightSwitch.operate();**

**return 0;**

**}**

**Solution:**

**Dependency Inversion Principle:**

class Switchable {

public:

virtual void turnOn() = 0;

virtual void turnOff() = 0;

};

class LightBulb : public Switchable {

public:

void turnOn() override {

std::cout << "Light bulb is on." << std::endl;

}

void turnOff() override {

std::cout << "Light bulb is off." << std::endl;

}

};

class LightSwitch {

public:

LightSwitch(Switchable& device) : device(device) {}

void operate() {

device.turnOn();

// ... additional logic ...

device.turnOff();

}

private:

Switchable& device;

};

int main() {

LightBulb bulb;

LightSwitch lightSwitch(bulb);

lightSwitch.operate();

return 0;

}

**Q5)**

**class Footballer {**

**private:**

**string name;**

**int age;**

**string role;**

**public:**

**Footballer(string name, int age, string role) {**

**this->name = name;**

**this->age = age;**

**this->role = role;**

**}**

**void getFootballerRole() {**

**switch (role) {**

**case 'g':**

**cout << "goalkeeper";**

**break;**

**case 'd':**

**cout << "defender";**

**break;**

**case 'm':**

**cout << "midfielder";**

**break;**

**case 'f':**

**cout << "forward";**

**break;**

**default:**

**throw runtime\_error("Unsupported footballer role: " + role);**

**}**

**}**

**};**

**int main() {**

**Footballer kante("Ngolo Kante", 31, "midfielder");**

**Footballer hazard("Eden Hazard", 32, "forward");**

**kante.getFootballerRole();**

**hazard.getFootballerRole();**

**return 0;**

**}**

**Solution:**

**Open Closed Principle:**

class PlayerRole {

public:

string getRole() = 0;

};

class GoalkeeperRole : public PlayerRole {

public:

string getRole() override {

return "goalkeeper";

}

};

class DefenderRole : public PlayerRole {

public:

string getRole() override {

return "defender";

}

};

class MidfieldRole : public PlayerRole {

public:

string getRole() override {

return "midfielder";

}

};

class ForwardRole : public PlayerRole {

public:

string getRole() override {

return "forward";

}

};

class Footballer {

private:

string name;

int age;

PlayerRole\* role;

public:

Footballer(string name, int age, PlayerRole\* role) {

this->name = name;

this->age = age;

this->role = role;

}

string getRole() {

return role->getRole();

}

};

int main() {

ForwardRole forwardRole;

Footballer hazard("Hazard", 32, &forwardRole);

cout << hazard.name << " plays in the " << hazard.getRole() << " line" <<endl;

MidfieldRole midfieldRole;

Footballer kante("Ngolo Kante", 31, &midfieldRole);

cout << kante.name << " is the best " << kante.getRole() << " in the world!" <<endl;

return 0;

}