MẪU STATE

*1.1. **Nêu vấn đề***

Giả sử chúng ta đang xây dựng một máy ATM (ATM machine) cho một ngân hàng. Máy ATM sẽ có một số trạng thái cơ bản, bao gồm: nhập mã PIN, kiểm tra mã PIN, rút tiền, gửi tiền (khách hàng gửi tiền vào), và hết tiền. Khi khách hàng sử dụng máy ATM, họ sẽ nhập mã PIN và yêu cầu kiểm tra xem mã PIN có đúng hay không. Sau đó, khách hàng có thể yêu cầu rút hoặc gửi tiền. Máy ATM sẽ ở trạng thái "hết tiền" khi không còn tiền trong máy.

*1.2. **Giải pháp đơn giản (Không sử dụng Design Pattern)***

Chúng ta sử dụng một lớp ATM để lưu trữ trạng thái hiện tại của máy ATM và các phương thức để thực hiện các hành động tương ứng với mỗi trạng thái.

*1.3. **Các vấn đề xảy ra với giải pháp đơn giản***

- Không linh hoạt: Giải pháp đã nêu trên không cung cấp tính linh hoạt trong việc thêm hoặc loại bỏ các trạng thái mới. Nếu muốn thêm hoặc loại bỏ một trạng thái, cần phải sửa đổi mã của phương thức xử lý trạng thái tương ứng.

- Sự trùng lặp: Các phương thức xử lý trạng thái có thể có một số mã lệnh giống nhau. Điều này có thể dẫn đến sự trùng lặp mã lệnh trong mã của chương trình, dẫn đến việc giảm tính tái sử dụng và khó khăn trong việc bảo trì.

- Khó khăn trong việc quản lý trạng thái: Việc quản lý các trạng thái sẽ trở nên phức tạp hơn khi số lượng các trạng thái tăng lên. Các phương thức xử lý trạng thái có thể bị phân tán trong nhiều tệp khác nhau, điều này có thể dẫn đến khó khăn trong việc tìm kiếm và bảo trì mã.

- Không tối ưu hóa được: Giải pháp này không cung cấp tính tối ưu hóa trong việc xử lý các trạng thái. Mỗi khi máy ATM chuyển đổi trạng thái, nó phải chạy một phương thức xử lý trạng thái khác nhau. Điều này có thể làm cho máy ATM hoạt động chậm hơn nếu số lượng các trạng thái lớn.

*1.4. **State Pattern***

State Pattern là một mẫu thiết kế thuộc nhóm Behavioral Pattern – những mẫu thiết kế xác định các mẫu giao tiếp chung giữa các object. Từ đó các mẫu này tăng tính linh hoạt trong việc thực hiện việc giao tiếp giữa các object.

State pattern cho rằng nên tạo các class mới cho tất cả các trạng thái có thể có của một object và trích xuất tất cả các hành vi dành riêng cho trạng thái vào các class đó.

Thay vì tự thực hiện tất cả các hành vi, đối tượng ban đầu, được gọi là ngữ cảnh, lưu trữ một tham chiếu đến một trong các đối tượng trạng thái đại diện cho trạng thái hiện tại của nó và ủy quyền tất cả các công việc liên quan đến trạng thái cho đối tượng đó.

Các thành phần trong State Pattern:

- Context: Là lớp có nhiều trạng thái, hành vi lớp sẽ bị thay đổi bởi trạng thái. Được sử dụng bởi Client. Client không truy cập trực tiếp đến State của Object. Lớp Context này chứa thông tin của ConcreteState object, cho hành vi nào tương ứng với trạng thái nào hiện đang được thực hiện

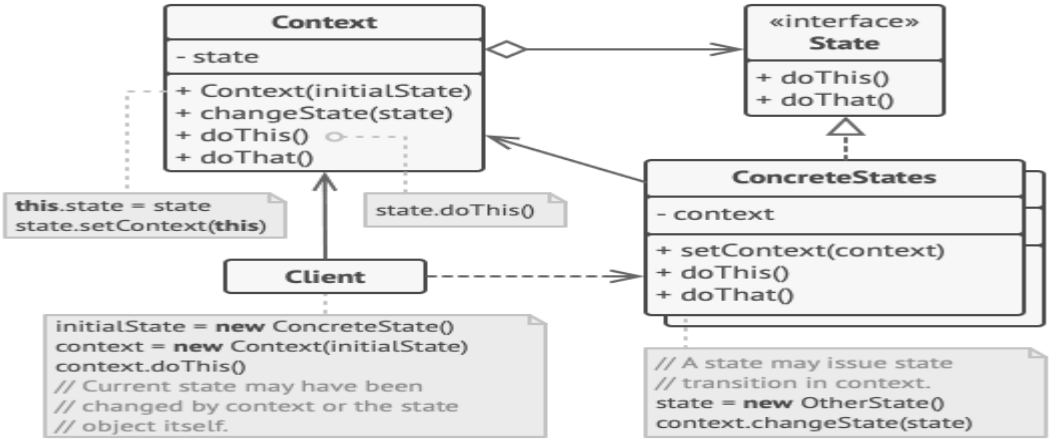
- State Interface: Là interface hoặc abstract class xác định các đặc tính cơ bản của tất cả ConcreteState Object. Chúng sẽ được sử dụng bởi đối tượng Context để truy cập chức năng có thể thay đổi.

- Concrete States: Là lớp cụ thể của state ứng với từng trạng thái của context

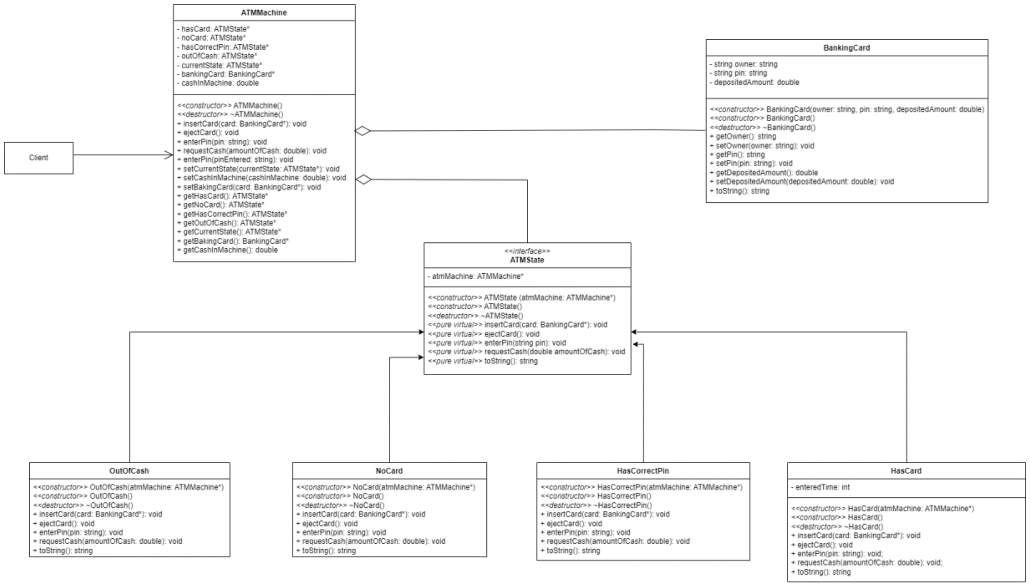
Cả Context và ConcreteState đều có thể thiết lập trạng thái tiếp theo của ngữ cảnh và thực hiện chuyển đổi trạng thái thực tế bằng cách thay thế đối tượng trạng thái được liên kết với ngữ cảnh.

Nhìn chung, mẫu thiết kế trạng thái có thể giúp quản lý độ phức tạp và giữ cho mã có thể duy trì được bằng cách cung cấp một cấu trúc rõ ràng để quản lý các trạng thái khác nhau của một đối tượng.

*1.5. **Sơ đồ lớp chung***



*1.6. **Sơ đồ lớp cho vấn đề ở bước 1***



*1.7. **Cài đặt State Pattern***

Các state của máy ATM bao gồm: **HasCard** (nhập mã pin), **OutOfCash** (hết tiền trong máy), **NoCard** (không có thẻ trong máy), **HasCorrectPin** (kiểm tra mã PIN).

ATM State là interface chung của các State : HasCard, OutOfCash, NoCard, HasCorrectPin.

Chúng ta sẽ sử dụng object để đóng gói state lại. Sau đó, việc chúng ta cần làm là sử dụng một con trỏ ATMState (currentState) trỏ đúng vào cái state object tương ứng với trạng thái hiện tại và call các method thông qua con trỏ đó.

Mỗi state object đều cần phải có các methods: InsertCard, EjectCard, EnterPin, RequestCash, nhưng tất nhiên là mỗi state sẽ xử lý theo các cách khác nhau.

Implementation of code

Class ATMState:

```
ATMState(ATMMachine* atmMachine);

ATMState();

~ATMState();

virtual void insertCard(BankingCard* card) = 0;

virtual void ejectCard() = 0;

virtual void enterPin(string pin) = 0;
```

```
virtual void requestCash(double amountOfCash) = 0;

virtual string toString() = 0;
```

Class **BankingCard**: Thông tin thẻ

```
BankingCard::BankingCard(string owner, string pin, double depositedAmount) {
    this->owner = owner;
    this->pin = pin;
    this->depositedAmount = depositedAmount;
}

BankingCard::BankingCard()
{
}

BankingCard::~BankingCard()
{
}

string BankingCard::getOwner()
{
    return owner;
}

void BankingCard::setOwner(string owner) {
    this->owner = owner;
}

string BankingCard::getPin() {
    return pin;
}
```

```
void BankingCard::setPin(string pin) {
    this->pin = pin;
}

double BankingCard::getDepositedAmount() {
    return depositedAmount;
}

void BankingCard::setDepositedAmount(double depositedAmount) {
    this->depositedAmount = depositedAmount;
}

string BankingCard::toString() {
    string temp;
    temp.append("BankingCard{");
    temp.append("owner='" + owner + '\'');
    temp.append(", pin='" + pin + '\'');
    temp.append(", depositedAmount=" + to_string(depositedAmount) + '}');
    return temp;
}
```

Class **HasCard** kế thừa từ class **ATMState:**

```
HasCard::HasCard(ATMMachine* atmMachine) : ATMState(atmMachine) {
    this->enteredTime = 0;
}

HasCard::HasCard()
{
}

HasCard::~HasCard()
```

```
{

}


void HasCard::insertCard(BankingCard* card)

{

    cout << "You already entered card\n";

}


void HasCard::ejectCard()

{

    atmMachine->setCurrentState(atmMachine->getNoCard());

    cout << "Your card is ejected\n";

}


void HasCard::enterPin(string pin)

{

    if (atmMachine->getBakingCard()->getPin() == pin)

    {

        cout << "Pin entered correctly\n";

        atmMachine->setCurrentState(atmMachine->getHasCorrectPin());

    }

    else

    {

        if (enteredTime < 3)

        {

            cout << "Incorrect pin, please enter your pin again\n";

            enteredTime++;

        }

        else

        {

            ejectCard();

        }
```

```
    }
}

void HasCard::requestCash(double amountOfCash)
{
    cout << "Please enter your pin\n";
}

string HasCard::toString()
{
    return "In has card state";
}
```

Class **OutOfCash** kế thừa từ class **ATMState:**

```
OutOfCash::OutOfCash(ATMMachine* atmMachine)
    : ATMState(atmMachine) {}

OutOfCash::OutOfCash()
{
}

OutOfCash::~OutOfCash()
{
}

void OutOfCash::insertCard(BankingCard* card) {
    cout << "We don't have any money" << endl;
    cout << "Your card is ejected" << endl;
}

void OutOfCash::ejectCard() {
```

```
    cout << "We don't have any money" << endl;
    cout << "Your card is ejected" << endl;
}

void OutOfCash::enterPin(string pin) {
    cout << "We don't have any money" << endl;
}

void OutOfCash::requestCash(double amountOfCash) {
    cout << "We don't have any money" << endl;
}

string OutOfCash::toString() {
    return "In out of cash state";
}
```

Class **NoCard** kế thừa từ class **ATMState**:

```
NoCard::NoCard(ATMMachine* atmMachine)
    : ATMState(atmMachine) {}

NoCard::NoCard() : ATMState(NULL)
{
}

NoCard::~NoCard()
{
}

void NoCard::insertCard(BankingCard* card)
{
    atmMachine->setBakingCard(card);
```

```
    atmMachine->setCurrentState(atmMachine->getHasCard());
    cout << "Your card is inserted\n";
}

void NoCard::ejectCard()
{
    cout << "ATM has no card to eject\n";
}

void NoCard::enterPin(string pin)
{
    cout << "You have not entered your card\n";
}

void NoCard::requestCash(double amountOfCash)
{
    cout << "You have not entered your card\n";
}

string NoCard::toString()
{
    return "In no card state";
}
```

Class **HasCorrectPin** kế thừa class **ATMState**:

```
HasCorrectPin::HasCorrectPin(ATMMachine* atmMachine)
    : ATMState(atmMachine) {}

HasCorrectPin::HasCorrectPin()
    : ATMState(NULL)
{
```

```
}

HasCorrectPin::~HasCorrectPin()
{
}

void HasCorrectPin::insertCard(BankingCard* card)
{
    cout << "You already enter your card\n";
}

void HasCorrectPin::ejectCard()
{
    cout << "Your card is ejected\n";
    atmMachine->setCurrentState(atmMachine->getNoCard());
}

void HasCorrectPin::enterPin(std::string pin)
{
    cout << "You already enter pin\n";
}

void HasCorrectPin::requestCash(double amountOfCash)
{
    if (amountOfCash > atmMachine->getCashInMachine())
    {
        cout << "We don't have that much money available\n";
        ejectCard();
    }
    else if (amountOfCash > atmMachine->getBakingCard()->getDepositedAmount())
```

```
        {
            cout << "Your account doesn't have enough money\n";
            ejectCard();
        }
        else
        {
            cout << amountOfCash << " is provided\n";
            atmMachine->setCashInMachine(atmMachine->getCashInMachine() -
amountOfCash);
            BankingCard* card = atmMachine->getBakingCard();
            card->setDepositedAmount(card->getDepositedAmount() - amountOfCash);
            ejectCard();
            if (atmMachine->getCashInMachine() < 0)
            {
                atmMachine->setCurrentState(atmMachine->getOutOfCash());
            }
        }
    }

    string HasCorrectPin::toString()
    {
        return "In has correct pin state";
    }
```

Class **ATMMachine** :

Trong constructor sẽ tạo ra instance tương ứng với 4 state: HasCard, NoCard, HasCorrectPin, OutOfCash. State hiện tại của ATMMachine được lưu bởi biến con trỏ currentState và được khởi tạo trỏ đến instance của state NoCard.

```
    ATMMachine::ATMMachine() {
        hasCard = new HasCard(this);
        noCard = new NoCard(this);
        hasCorrectPin = new HasCorrectPin(this);
```

```
    outOfCash = new OutOfCash(this);
    bankingCard = nullptr;
    currentState = noCard;
    cashInMachine = 100000;
    if (cashInMachine < 0) {
        currentState = outOfCash;
    }
}

ATMMachine::~ATMMachine()
{
}

void ATMMachine::insertCard(class BankingCard* card) {
    currentState->insertCard(card);
}

void ATMMachine::ejectCard() {
    currentState->ejectCard();
}

void ATMMachine::requestCash(double amountOfCash) {
    currentState->requestCash(amountOfCash);
}

void ATMMachine::enterPin(string pinEntered) {
    currentState->enterPin(pinEntered);
}

void ATMMachine::setCurrentState(ATMState* currentState) {
    this->currentState = currentState;
}
```

```
void ATMMachine::setCashInMachine(double cashInMachine) {
    this->cashInMachine = cashInMachine;
}

void ATMMachine::setBakingCard(class BankingCard* card) {
    this->bankingCard = card;
}

ATMState* ATMMachine::getHasCard() {
    return hasCard;
}

ATMState* ATMMachine::getNoCard() {
    return noCard;
}

ATMState* ATMMachine::getHasCorrectPin() {
    return hasCorrectPin;
}

ATMState* ATMMachine::getOutOfCash() {
    return outOfCash;
}

ATMState* ATMMachine::getCurrentState() {
    return currentState;
}

class BankingCard* ATMMachine::getBakingCard() {
    return bankingCard;
}
```

```
double ATMMachine::getCashInMachine() {
    return cashInMachine;
}
```

Kết quả :

```
1.Insert card
2.Eject card
3.Enter pin
4.Withdraw money

Your choice: 1

Your card is inserted
In has card state
```

Thêm một thẻ vào ATM

```
1.Insert card
2.Eject card
3.Enter pin
4.Withdraw money

Your choice: 1

Your card is inserted
In has card state
```

Nhập và Kiểm tra mã Pin

```
1.Insert card
2.Eject card
3.Enter pin
4.Withdraw money

Your choice: 4

Enter amount of money you want to get: 100000
100000 is provided
Your card is ejected
In no card state
```

Rút tiền , Rút thẻ

Có thể thấy rằng code bây giờ đã tách biệt, các state được đóng gói ra các class khác nhau, đồng thời dễ bảo trì và mở rộng hơn so với trường hợp cài đặt theo cách thông thường.

1.8. ***Ưu điểm và Nhược điểm của State Pattern***

Ưu điểm của State Pattern:

- Đảm bảo nguyên tắc Single Responsibility (SRP): Tách biệt mỗi State tương ứng với 1 class riêng biệt.

- Đảm bảo nguyên tắc Open/Closed Principle (OCP): chúng ta có thể thêm một State mới mà không ảnh hưởng đến State khác hay Context hiện có.

- Giữ hành vi cụ thể tương ứng với mỗi State (trạng thái).

- Giúp chuyển State một cách rõ ràng.

- Loại bỏ các câu lệnh xét trường hợp (If, Switch case) giúp đơn giản code của context.

Nhược điểm của State Pattern:

- Việc sử dụng state pattern có thể quá mức cần thiết nếu state machine chỉ có một vài trạng thái hoặc hiếm khi thay đổi có thể dẫn đến việc tăng độ phức tạp của code.

- Các đối tượng trạng thái có thể trở nên liên kết chặt chẽ với đối tượng Context, gây khó khăn cho việc di chuyển chúng sang các Context khác hoặc sử dụng lại chúng trong các ứng dụng khác.

1.9. ***Các vấn đề thực tế khác và cách áp dụng Flyweight Design Pattern trong web dev, mobile dev.***

- Xây dựng một máy bán hàng tự động cần phân phối các sản phẩm khác nhau dựa trên trạng thái hiện tại của nó (ví dụ: có đủ hàng, hết hàng, cần bảo trì,…).

- Xây dựng một trình chỉnh sửa tài liệu cần thay đổi hành vi của nó dựa trên trạng thái hiện tại (ví dụ: chỉ đọc, chỉnh sửa, lưu, …).

Ứng dụng trong web dev, mobile dev:

- Quản lý trạng thái của ứng dụng: Trong các ứng dụng Web và App, thường có nhiều trạng thái khác nhau, ví dụ như đang chờ xử lý, đã hoàn thành, hoặc lỗi. State Pattern có thể được sử dụng để quản lý trạng thái của các đối tượng trong ứng dụng, giúp cho các đối tượng phản hồi đúng cách với các trạng thái khác nhau.

- Quản lý việc chuyển đổi trạng thái: Trong một số trường hợp, các đối tượng có thể chuyển đổi giữa các trạng thái khác nhau. State Pattern có thể được sử dụng để quản lý quá trình chuyển đổi trạng thái của các đối tượng và đảm bảo rằng các đối tượng phản hồi đúng cách với các trạng thái khác nhau.

- Quản lý quy trình công việc (workflow): Trong các ứng dụng quản lý công việc, State Pattern có thể được sử dụng để quản lý các quy trình công việc khác nhau. Ví dụ, nếu một công việc đang ở trạng thái "đang chờ xử lý", nó sẽ phản hồi khác với khi nó đang ở trạng thái "đang thực hiện".

- Quản lý trạng thái của giao diện người dùng: Trong các ứng dụng Web và App, giao diện người dùng thường phải phản hồi với các trạng thái khác nhau, ví dụ như khi button được nhấn, hoặc khi kết nối Internet bị mất. State Pattern có thể được sử dụng để quản lý trạng thái của giao diện người dùng và đảm bảo rằng giao diện người dùng phản hồi đúng cách với các trạng thái khác nhau.

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN TP.HCM

KHOA CÔNG NGHỆ THÔNG TIN

BTC ÔN THI HỌC KỲ KHÓA 2016

Bài giải tham khảo

Đề thi cuối kì LTHĐT các năm 2014, 2015, 2016

- Người giải: Vương Hy – Lớp 16CNTN

Cập nhật: 25/12/2017

Đề 2016

Câu 1:

Ellipse ellipse(2.0/3, 6);

⇨ Gọi contructor của lớp Ellipse nhưng => constructor của Ellipse gọi constructor của lớp cha là Shape

construct Shape

construct Ellipse

cout << "ellipse: " << endl;

ellipse:

cout << ellipse.Description() << endl;

In số thực thì sẽ in tối đa 6 chữ số có nghĩa (kể cả trước và sau dấu thập phân)

Do đã gọi phương thức shape->Scale(1.0/2); nên kích thước thay đổi

Ellipse(a=0.333333, b= 3)

cout << ellipse.Area() << endl;

3.14159

cout << ellipse.InterfaceType().name() << endl;

Do biến ellipse có kiểu Ellipse nên sẽ gọi hàm trong lớp Ellipse và in ra

Ellipse

cout << ellipse.ImplementationType().name() << endl;

Dù hàm được gọi là hàm của lớp Shape nhưng con trỏ this lúc này sẽ chứa đối tượng Ellipse nên sẽ in ra là Ellipse (tùy trình biên dịch như trong vs in ra là class Ellipse)

Ellipse

cout << "shape = &ellipse: " << endl;

shape = &ellipse:

cout << shape->Description() << endl;

Do hàm Description khai báo là hàm ảo (virtual) nên hàm Description của lớp Ellipse sẽ được gọi

Ellipse(a=0.333333, b= 3)

cout << shape->Area() << endl;

3.14159

cout << shape->InterfaceType().name() << endl; Do lớp phương thức InterfaceType không phải là hàm ảo nên phương thức của lớp Shape sẽ được gọi

Shape

cout << shape->ImplementationType().name() << endl; typeid sẽ trả về đúng kiểu đối tượng mà con trỏ this này đang trỏ tới

Ellipse

Hết hàm main => gọi destructor của các biến cục bộ theo thứ tự ngược lại với thứ tự khai báo. Ở đây chỉ có biến ellipse Destructor sẽ được gọi theo thứ tự từ lớp con đến lớp cha

destruct Ellipse

destruct Shape

Tóm lại output là:

construct Shape

construct Ellipse

ellipse:

Ellipse(a=0.333333, b=3)

3.14159

Ellipse

Ellipse

shape = &ellipse:

Ellipse(a=0.333333, b=3)

3.14159

Shape

Ellipse

destruct Ellipse

destruct Shape

- Shape::Description() và Circle::Description()
    - Đây là ghi đè (overriding) hàm, Circle::Description() override lại hàm Shape::Description() của lớp cha vì Description ở lớp Circle khác với lớp Shape nên ta không thể dùng lại phương thức của lớp cha hơn nữa đây lại là hàm ảo nên việc override này sẽ cho phép đối tượng Circle được quản lý bằng biến kiểu Shape * nhưng vẫn dùng phương thức Description của lớp Circle
- Ellipse::Scale(float) và Ellipse::Scale(float, float)

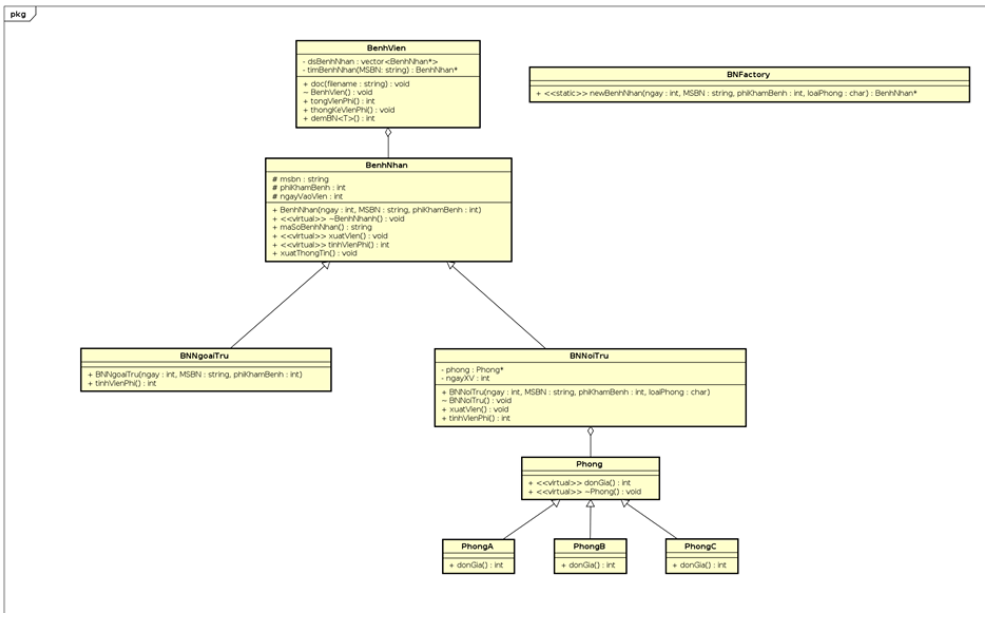
- Đây là nạp chồng, quá tải (overload) hàm, cùng một tên hàm nhưng lại có tham số truyền vào khác nhau thường overload hàm sẽ giúp ta có nhiều cách để thực hiện cùng một công việc cho nhiều kiểu tham số khác nhau. Như bài này Scale(float) chỉ có 1 tham số vào và sẽ thay đổi theo cả phương cùng một tỉ lệ, còn Phương thức Scale(float, float) sẽ có thể thay đổi trục chính và trục phụ theo 2 tỉ lệ riêng biệt.

- Shape::InterfaceType() và Circle::InterfaceType()
  - Đây cũng là ghi đè hàm (overriding). Nhưng lần này 2 phương thức này không phải là hàm ảo nên biến có kiểu nào sẽ gọi đúng phương thức ở lớp đó được xác định lúc dịch chương trình.
- Circle::InterfaceType() và Ellipse::InterfaceType()
  - Không có quan hệ gì

c) Dòng 88: Lỗi lớp Shape không có phương thức Scale nhận vào 2 tham số float
Dòng 102 + 103: Lớp Shape có chứa hàm thuần ảo (Area, Scale) nên là lớp thuần ảo do đó không thể tạo đối tượng có kiểu Shape
d) Dòng 88: ép kiểu động bằng cách (dynamic_cast<Ellipse *>shape)->Shape(3, 1.0/3)
Dòng 102, 103, có thể gọi tường minh trực tiếp shape->Shape::Description();

Câu 3:

Sơ đồ lớp:



Cài đặt

```
class BenhVien {
private:
	vector<BenhNhan*> dsBenhNhan;
	BenhNhan* timBenhNhan(string MSBN) {
		for (auto& x: dsBenhNhan)
			if (x->maSoBenhNhan() == MSBN)
				return x;
		return NULL;
	}
public:
	void doc(string fileName) {
		//cout << "asdfjasfajsgdfasdf";
```

```
		freopen(fileName.c_str(), "r", stdin);
		stringstream ss;
		string s;
		int ngay;
		while (cin >> ngay) {
			// cout << s << endl;
			int phiKhamBenh;
			string MSBN;
			string type;
			char loaiPhong = 0;
			cin >> MSBN >> type;
			if (type == "XV") {
				timBenhNhan(MSBN)->xuatVien(ngay);
				continue;
			} else if (type == "TKVP") {
				for (auto& x: dsBenhNhan)
					x->xuatVien(ngay);
				continue;
			}

			cin >> phiKhamBenh;

			if (type == "NV")
				cin >> loaiPhong;
			dsBenhNhan.push_back(BNFactory::newBenhNhan(ngay, MSBN, phiKhamBenh,
loaiPhong));
		}
	}
	~BenhVien() {
		for (BenhNhan*& x: dsBenhNhan)
			delete x;
	}
	int tongVienPhi() {
		int res = 0;
		for (BenhNhan*& x: dsBenhNhan)
			res += x->tinhVienPhi();
		return res;
	}
	void thongKeVienPhi() {
		cout << "Thong tin benh nhan:\n";
		for (BenhNhan*& x: dsBenhNhan) {
			x->xuatThongTin();
			cout << endl;
		}
		cout << "> Tong ket vien phi: " << tongVienPhi();
	}
	template<class T>
	int demBN() {
		int res = 0;
		for (auto& x: dsBenhNhan)
			res += !!dynamic_cast<T*>(x);
		return res;
	}
};

class BenhNhan {
protected:
	string MSBN;
```

```
	int phiKhamBenh;
	int ngayVaoVien;
public:
	BenhNhan(int ngay, string _MSBN, int _phiKhamBenh): ngayVaoVien(ngay),
MSBN(_MSBN), phiKhamBenh(_phiKhamBenh) {}
	virtual ~BenhNhan() {}
	string maSoBenhNhan() {
		return MSBN;
	}
	virtual void xuatVien(int) {}
	virtual int tinhVienPhi() {return 0;}
	void xuatThongTin() {
		cout << "Ma so benh nhan: " << MSBN << endl;
		cout << "Vien phi: " << tinhVienPhi();
	}
};

class BNNoiTru: public BenhNhan {
	Phong* phong;
	int ngayXV = -1;
public:
	BNNoiTru(int ngay, string _MSBN, int _phiKhamBenh, char loaiPhong): BenhNhan(ngay,
_MSBN, _phiKhamBenh) {
		ngayVaoVien = ngay;
		if (loaiPhong == 'A')
			phong = new PhongA;
		else if (loaiPhong == 'B')
			phong = new PhongB;
		else
			phong = new PhongC;
	}
	~BNNoiTru() {
		delete phong;
	}
	void xuatVien(int ngay) {
		if (ngayXV == -1)
			ngayXV = ngay;
	}
	int tinhVienPhi() {
		return (ngayXV - ngayVaoVien) * (phong->donGia() + phiKhamBenh) ;
	}
};

class BNNgoaiTru: public BenhNhan {
public:
	BNNgoaiTru(int ngay, string _MSBN, int _phiKhamBenh): BenhNhan(ngay, _MSBN,
_phiKhamBenh) {}
	~BNNgoaiTru() {}
	int tinhVienPhi() {
		return phiKhamBenh;
	}
};
class BNFactory {
public:
	static BenhNhan* newBenhNhan(int ngay, string MSBN, int phiKhamBenh, char
loaiPhong = 0) {
		if (loaiPhong)
			return new BNNoiTru(ngay, MSBN, phiKhamBenh, loaiPhong);
```

```
                else
                        return new BNNgoaiTru(ngay, MSBN, phiKhamBenh);
        }
};

class Phong {
public:
        virtual int donGia() = 0;
        virtual ~Phong() {}
};
class PhongC: public Phong {
public:
        int donGia() {
                return 600000;
        }
};
class PhongB: public Phong {
public:
        int donGia() {
                return 900000;
        }
};
class PhongA: public Phong {
public:
        int donGia() {
                return 1400000;
        }
};
```

Đề 2015

Câu 1:

a.

Kết quả in ra là

3 4

Giải thích:

Ở a.f(3) dù f là hàm ảo nhưng do A không phải là kiểu con trỏ nên chương trình vẫn sẽ gọi hàm f của A

Ở b.f(3) thì chương trình sẽ gọi hàm B::f (3). Hàm B::f(3) đó gọi hàm A::f(3 + 1) in ra 4

b.

Lớp A có cấp phát bộ nhớ và sử dụng biến con trỏ int * a nên phải thực hiện cài đặt 3 ông lớn là copy constructor, operator =, và destructor. Trong đoạn mã đề bài cho chỉ mới cài đặt destructor nên ta sẽ cài đặt thêm

```
class A {
public:
	A() { a = new int[3]; for (int i = 0; i < 3; i++) a[i] = i + 1; }
	~A() { delete[] a; }

	A(const A & obj) {
		a = new int[3];
		for (int i = 0; i < 3; i++) a[i] = obj.a[i];
	}

	A & operator =(const A & obj) {
		for (int i = 0; i < 3; i++) a[i] = obj.a[i];
		return (*this);
	}
private:
	int *a;
};
```

c)

```
class Singleton {
private:
	Singleton * obj = NULL;
	Singleton();
public:
	static Singleton * getObj() {
		if (obj == NULL)
			obj = new Singleton();

		return obj;
	};
};
```

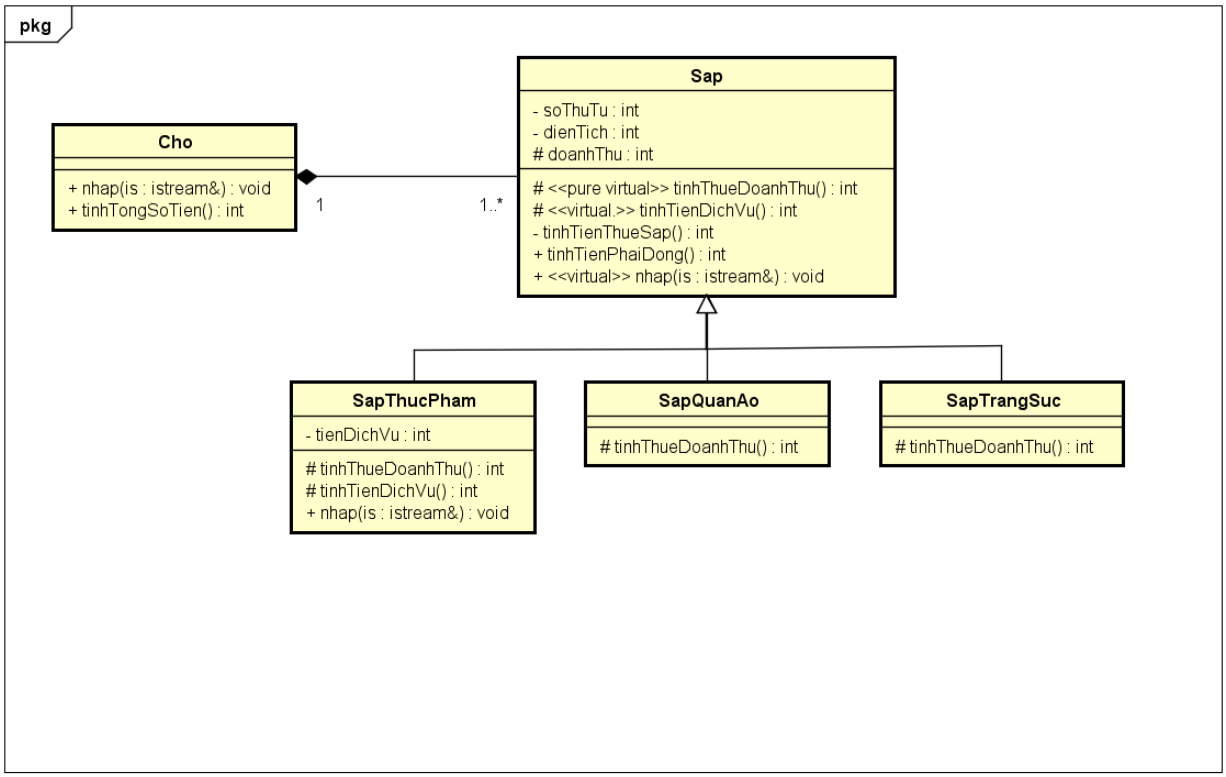
Lớp Singleton chỉ có thể tạo được một thể hiện duy nhất do ta không thể tạo ra lớp Singleton qua việc gọi contructor hay khai báo biến (do constructor là phương thức private). Ta chỉ có thể tạo ra đối tượng Singleton qua việc gọi Singleton::getObj() nhờ đó ta có thể quản lý số lượng đối tượng Singleton được tạo ra. Nếu đã có một đối tượng được tạo ra rồi thì ta sẽ không tạo ra nữa mà trả về đối tượng đó

Câu 2:

```
class BigInteger {
private:
	string number;
public:
	BigInteger(string number = "0") : number(number) {
	}

	bool operator == (const BigInteger & x) const {
		return number == x.number;
	}

	friend istream & operator >> (istream & is, BigInteger & number);
};

istream & operator >> (istream & is, BigInteger & number) {
	return is >> number;
}
```

Câu 3:

Sơ đồ lớp



Cài đặt:

```
#include <iostream>
#include <vector>
```

```
using namespace std;

class Sap {
private:
	const int DON_GIA_SAP = 40000000;

	int soThuTu;
	int dienTich;
	int tinhTienThueSap() {
		return DON_GIA_SAP * dienTich;
	};

protected:
	int doanhThu;
	virtual int tinhThueDoanhThu() = 0;
	virtual int tinhTienDichVu() {
		return 0;
	}
public:
	int tinhTienPhaiDong() {
		return tinhTienThueSap() + tinhTienDichVu() + tinhThueDoanhThu();
	}
	virtual void nhap(istream & is) {
		cout << "Nhap so thu tu: ";
		is >> soThuTu;
		cout << "Nhap dien tich: ";
		is >> dienTich;
		cout << "Nhap doanh thu: ";
		is >> doanhThu;
	}
};

class SapThucPham : public Sap {
private:
	int tienDichVu;
protected:
	int tinhThueDoanhThu() {
		return doanhThu * 5 / 100;
	}
	int tinhTienDichVu() {
		return tienDichVu;
	}
};


class SapQuanAo : public Sap {
protected:
	int tinhThueDoanhThu() {
		return doanhThu * 10 / 100;
	}
};

class SapTrangSuc : public Sap {
protected:
	int tinhThueDoanhThu() {
		if (doanhThu >= 100000000)
			return doanhThu * 30 / 100;
```

```
		else
			return doanhThu * 20 / 100;
	}
};

class Cho {
private:
	vector<Sap *> dsSap;
public:
	void nhap(istream & is) {
		int n;
		cout << "Nhap so luong sap: ";
		is >> n;
		dsSap.resize(n);

		for (int i = 0; i < n; i++) {
			cout << "Nhap loai 1 - sap thuc pham, 2 - sap quan ao, 3 - sap trang
suc: ";

			int loai;
			is >> loai;

			switch (loai) {
			case 1:
				dsSap[i] = new SapThucPham();
				break;
			case 2:
				dsSap[i] = new SapQuanAo();
				break;
			case 3:
				dsSap[i] = new SapTrangSuc();
				break;
			default:
				cout << "Loai sap khong hop le" << endl;
			}

			dsSap[i]->nhap(is);
		}
	};

	int tinhTongSoTien() {
		int total = 0;

		for (int i = 0; i < dsSap.size(); i++)
			total += dsSap[i]->tinhTienPhaiDong();

		return total;
	}
};
```

Đề 2014

Câu 1

a.

Cần cài đặt thêm 3 ông lớn và giá trị mặc định cho hiệu xe

```
Bike() : brand(NULL) {
	set_brand("default");
}

Bike(const Bike & b) : brand(NULL) {
	set_brand(b.brand);
}

virtual ~Bike() {
	if (brand != NULL)
		delete[]brand;
}

Bike & operator = (const Bike & b) {
	set_brand(b.brand);
}

void set_brand(char *brand) {
	if (brand != NULL)
		delete[] brand;

	this->brand = new char[strlen(brand)+1];
	strcpy(this->brand, brand);
	//hoac this->brand = strdup(brand);
}
```

b.

Kết quả là

default:48 default:48

Do hàm display nhận tham số theo kiểu tham chiếu Bike& và move được khai báo hàm ảo nên hàm move của đúng đối tượng EBike sẽ được gọi

c.

Đối tượng là thể hiện của lớp. Một lớp có thể có nhiều đối tượng khác nhau. Mỗi đối tượng đều của 1 lớp sẽ có nhưng phương thức và thuộc tính mà lớp đó quy định nhưng giá trị của thuộc tính có thể khác nhau.

Lớp: Bike, EBike

Đối tượng được lưu trong các biến b1, b2.

Câu 2:

a.

```
class EyeFace : public Face {
private:
	int eyes;
public:
	EyeFace(string sh, int eyes) : Face(sh), eyes(eyes) {
	}
```

```
	virtual void show() {
		Face::show();
		cout << "Eyes: " << eyes << endl;
	}

	EyeFace * clone() {
		return new EyeFace(getShape(), eyes);
	}
};
```

b.

Hàm main không chạy được vì lớp Face là lớp thuần ảo (chưa cài đặt Phương thức clone của IFace) nên không thể tạo đối tượng Face nên ta sẽ cài thêm hàm clone. Tuy nhiên đối tượng fc vẫn không được tạo ra do lớp Face không có constructor mặc định nên ta sẽ cài đặt constructor mặt định.

```
	Face * clone() {
		return new Face(shape);
	}

	Face() :shape("NULL") {

	}
```

Khi đó kết quả xuất ra main là

Shape: Rectangle

Shape: Rectangle

Shape: Rectangle

The same 3 lines?

c.

Do khi clone thì đối tượng mới được tạo ra bằng operator new nhưng lại chưa được delete nên sẽ gây rò rỉ bộ nhớ nên ta sẽ phải delete sau khi dùng

```
void testFace(IFace* fc) {
	IFace* a[3] = { fc, fc->clone(), fc->clone() };
	for (int i = 0; i<3; i++) {
		a[i]->show();
	}
	cout << "The same 3 lines ? ";

	delete a[1];
	delete a[2];
}
```

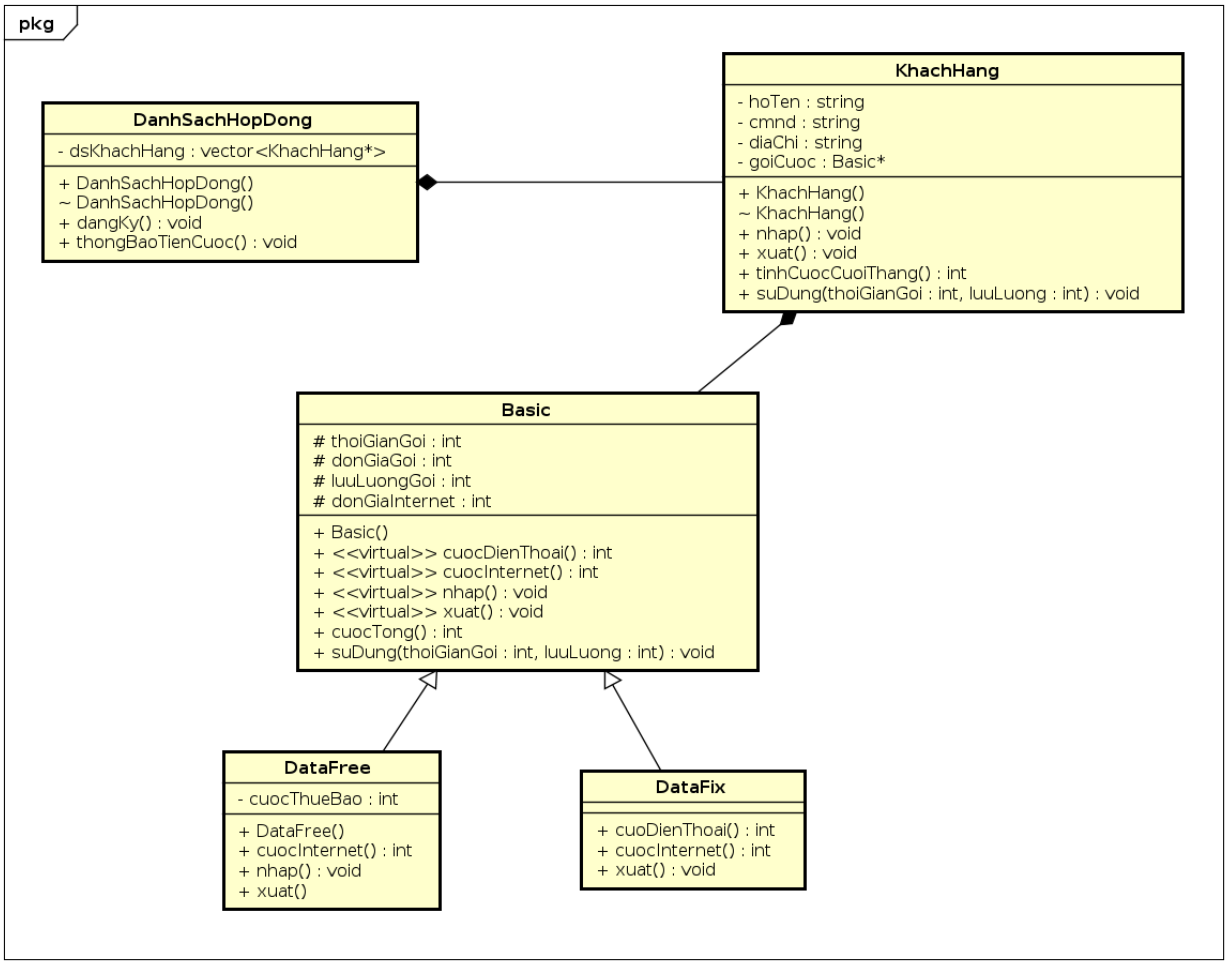
Chỉnh lại lớp EyeFace để thêm một biến static instanceCount để đếm số đối tượng được tạo ra của lớp EyeFace. Khi constructor được gọi thì sẽ tăng biến đó lên 1 và khi destructor được gọi thì sẽ giảm đi 1

```
class EyeFace : public Face {
private:
…
	static int instanceCount;
public:
	EyeFace(string sh, int eyes) : Face(sh), eyes(eyes) {
		instanceCount++;
	}
```

```
	~EyeFace() {
		instanceCount--;
	}

	static int countNumberOfInstance() {
		return instanceCount;
	}
…
};

int EyeFace::instanceCount = 0;


int main() {
…
	cout << "Number of EyeFace instance: " << EyeFace::countNumberOfInstance() <<
endl;
	return 0;
}
```

Câu 3:

Sơ đồ lớp



Cài đặt

```
class DanhSachHopDong {
	vector<KhachHang*> dsKhachHang;
public:
	void dangKy();
	void thongBaoTienCuoc();
	DanhSachHopDong();
	~DanhSachHopDong();
};

class KhachHang {
	string hoTen;
	string cmnd;
	string diaChi;
	Basic* goiCuoc;
public:
	KhachHang();
	~KhachHang();
	void nhap();
	void xuat();
	int tinhCuocCuoithang();
	void suDung(int, int);
};

class Basic {
protected:
	int thoiGianGoi;
	int donGiaGoi;
	int luuLuong;
	int donGiaInternet;
public:
	virtual int cuocDienThoai();
	virtual int cuocInternet();
	virtual void nhap();
	virtual void xuat();
	int cuocTong();
	void suDung(int, int);
	Basic();
	virtual ~Basic();
};

class DataFix: public Basic {
public:
	int cuocDienThoai();
	int cuocInternet();
	void xuat();
	~DataFix();
};

class DataFree: public Basic {
	int cuocThueBao;
public:
	DataFree();
	int cuocInternet();
	void nhap();
	void xuat();
	~DataFree();
};
```

```
DanhSachHopDong::DanhSachHopDong() {}
DanhSachHopDong::~DanhSachHopDong() {
	for (int i = 0; i < dsKhachHang.size(); i++)
		delete dsKhachHang[i];
}
void DanhSachHopDong::dangKy() {
	int n;
	cout << "Nhap so luong khach hang: ";
	cin >> n;
	dsKhachHang.resize(n);
	for (int i = 0; i < n; i++) {
		cout << "Nhap thong tin khach hang thu " << i + 1 << ":\n";
		dsKhachHang[i] = new KhachHang;
		dsKhachHang[i]->nhap();
	}
}
void DanhSachHopDong::thongBaoTienCuoc() {
	cout << "So luong hop dong: " << dsKhachHang.size() << endl;
	for (int i = 0; i < dsKhachHang.size(); i++) {
		cout << "Thong tin hop dong thu " << i + 1 << ":\n";
		dsKhachHang[i]->xuat();
		cout << endl;
		cout << "Tien cuoc cuoi thang: " << dsKhachHang[i]->tinhCuocCuoithang() <<
endl;
	}
}

KhachHang::KhachHang() {}
KhachHang::~KhachHang() {
	delete goiCuoc;
}
void KhachHang::nhap() {
	cout << "Nhap ho ten: ";
	cin >> hoTen;
	cout << "Nhap cmnd: ";
	cin >> cmnd;
	cout << "Nhap dia chi: ";
	cin >> diaChi;
	int c;
	cout << "Nhap thong tin goi cuoc:\nChon loai goi cuoc (0: Basic, 1: DataFree, 2:
DataFix): ";
	cin >> c;
	if (c == 0)
		goiCuoc = new Basic;
	else if (c == 1)
		goiCuoc = new DataFree;
	else
		goiCuoc = new DataFix;
	goiCuoc->nhap();
}
void KhachHang::xuat() {
	cout << "Ho ten: " << hoTen << endl;
	cout << "CMND: " << cmnd << endl;
	cout << "Dia chi: " << diaChi << endl;
	cout << "Thong tin goi cuoc:\n";
	goiCuoc->xuat();
}
int KhachHang::tinhCuocCuoithang() {
```

```
	return goiCuoc->cuocTong();
}
void KhachHang::suDung(int thoiGianGoi, int luuLuong) {
	goiCuoc->suDung(thoiGianGoi, luuLuong);
}

Basic::Basic() {
	thoiGianGoi = 0;
	donGiaGoi = 1000;
	luuLuong = 0;
	donGiaInternet = 200;
}
Basic::~Basic() {}
int Basic::cuocDienThoai() {
	return thoiGianGoi * donGiaGoi;
}
int Basic::cuocInternet() {
	return luuLuong * donGiaInternet;
}
int Basic::cuocTong() {
	return 1.1 * (cuocInternet() + cuocDienThoai());
}
void Basic::nhap() {}
void Basic::xuat() {
	cout << "Ten goi cuoc: Basic";
}
void Basic::suDung(int _thoiGianGoi, int _luuLuong) {
	thoiGianGoi += _thoiGianGoi;
	luuLuong += _luuLuong;
}

DataFix::~DataFix() {}
int DataFix::cuocDienThoai() {
	return Basic::cuocDienThoai() * 0.9;
}
int DataFix::cuocInternet() {
	return 1000000;
}
void DataFix::xuat() {
	cout << "Ten goi cuoc: DataFix";
}

DataFree::DataFree() {}
DataFree::~DataFree() {}
int DataFree::cuocInternet() {
	return cuocThueBao + (luuLuong <= 0 ? 0: Basic::cuocInternet());
}
void DataFree::nhap() {
	cout << "Nhap cuoc thue bao: ";
	cin >> cuocThueBao;
	cout << "Nhap nguong luu luong mien phi: ";
	cin >> luuLuong;
	luuLuong *= -1;
}
void DataFree::xuat() {
	cout << "Ten goi cuoc: DataFree";
}
```

Câu 1: Cho đoạn mã nguồn sau: (Đã include đầy đủ các thư viện chuẩn cần thiết)

```
1  class Bike {
2  private:
3     char* brand; // hiệu xe
4  public:
5     virtual void move(int t1) {
6         cout << brand << ":" << t1*12 << " ";
7     }
8  };
9
10 class EBike : public Bike {
11     public: void move(int t2) {
12       Bike::move(t2*2);
13    }
14 };
```

```
15 void display(Bike& a, EBike& b) {
16    a.move(2);
17    b.move(2);
18 }
19
20 int main() {
21    EBike b1, b2;
22    display(b1, b2);
23    return 0;
24 }
```

a. Hãy cài đặt bổ sung các phương thức cần thiết để giải quyết các vấn đề về thuộc tính con trỏ cho lớp đối tượng Bike.
b. Cho biết kết quả xuất ra màn hình của chương trình trên (sau khi đã bổ sung code đầy đủ ở câu (a)). Giải thích ngắn gọn.
c. Nêu sự khác biệt giữa *lớp đối tượng (class)* và *đối tượng cụ thể (object, instance)*. Liệt kê các *lớp đối tượng* và các *đối tượng cụ thể* trong đoạn mã nguồn trên.

Câu 2: Cho đoạn mã nguồn sau: (Đã include đầy đủ các thư viện chuẩn cần thiết)

```
1  class IFace {
2  public:
3     virtual void show()=0;
4      virtual IFace* clone()=0;
5     virtual ~IFace() {
6     }
7  };
```

```
8  class Face : public IFace {
9  private:
10    string shape;
11 protected:
12    string getShape();
13 public:
14    Face(string sh) : shape(sh) {
15    }
16    virtual void show() {
17       cout << “Shape: ” << shape << endl;
18    }
19 };
```

a. Hãy viết lớp `EyedFace` có:
   - Hai thuộc tính: `shape` kiểu chuỗi và `eyes` kiểu số nguyên;
   - Hai phương thức: `show()` xuất ra màn hình giá trị của cả 2 thuộc tính, và `clone()` trả về một đối tượng `EyedFace` mới là một bản sao của đối tượng này;
   - Một hàm tạo (constructor) nhận 2 tham số tương ứng với 2 thuộc tính.

   Yêu cầu: Tái sử dụng tối đa mã nguồn đã cho.

b. Cho đoạn mã nguồn sau nối tiếp theo phần trên (cả đoạn cho sẵn lẫn của câu (a)):

```
1  void testFace(IFace* fc) {
2      IFace* a[3] = { fc, fc->clone(), fc->clone() };
3      for(int i=0; i<3; i++) {
4        a[i]->show();
5      }
6      cout << "The same 3 lines?";
7  }
```

```
8  int main() {
9     Face fc;
10    Face fc1("Rectangle");
11    testFace(&fc1);
12    return 0;
13 }
```

- Hàm `main()`có lỗi không thể chạy được. Hãy giải thích cụ thể lý do lỗi đó.
- Hãy sửa lỗi của hàm `main()`để chạy được hàm `testFace()` và cho biết kết quả xuất ra màn hình của hàm `main()`đó.

c. Hãy cải tiến chương trình bên trên (cả 2 câu (a) và (b)) như sau:

- Hàm `testFace()`quản lý bộ nhớ chưa hiệu quả. Hãy sửa lại lỗi bộ nhớ đó.
- Hãy viết thêm mã vào lớp `EyedFace` để đếm tổng số đối tượng (object, instance) của lớp đó và cập nhật mỗi khi tạo/huỷ đối tượng. Sau đó hãy thêm mã vào cuối hàm `main()`để kiểm tra xem còn bao nhiêu đối tượng `EyedFace` trong bộ nhớ.
  Lưu ý: Chỉ cần ghi phần mã bổ sung và tên lớp/hàm, còn lại thì ghi ba chấm.

Câu 3:

Một công ty viễn thông cung cấp dịch vụ điện thoại và internet hỗn hợp cho khách hàng theo hình thức trả sau. Để sử dụng dịch vụ, khách hàng cần ký hợp đồng với công ty. Trong hợp đồng, cần có các thông tin cá nhân của khách hàng (họ tên, chứng minh nhân dân, địa chỉ) và thông tin về cách tính cước mà khách hàng chọn. Cuối mỗi tháng, khách hàng sẽ được thông báo cước tùy theo lượng sử dụng của mình tương ứng với gói cước đã đăng ký ban đầu.

Hợp đồng với gói cước Basic có cách tính tiền như sau:

Cước điện thoại = *Thời gian gọi* (phút) * *Đơn giá gọi* (1000 đồng/phút).

Cước internet = *Lưu lượng truy cập* (MB) * *Đơn giá truy cập* (200 đồng/MB).

Cước tổng = *Cước điện thoại* + *Cước internet* + 10% VAT.

Để thu hút thêm đối tượng khách hàng thường xuyên sử dụng internet, công ty mở rộng thêm hai loại hợp đồng mới có cách tính cước linh hoạt như sau:

Gói cước	**Cước điện thoại**	**Cước internet**
Data Free	Tương tự gói Basic	• Nếu *Lưu lượng truy cập ≤ Ngưỡng lưu lượng miễn phí* ⇨ Chỉ đóng *Cước thuê bao*. • Nếu *Lưu lượng truy cập > Ngưỡng lưu lượng miễn phí* ⇨ *Cước thuê bao + Cước lưu lượng vượt ngưỡng*. <u>Ghi chú:</u> - *Cước thuê bao* và *Ngưỡng lưu lượng miễn phí* được công ty xác định lúc lập hợp đồng đăng ký cho khách hàng và được ghi trong mỗi hợp đồng: Có thể khác nhau tuỳ vào lúc lập hợp đồng nhưng không đổi sau đó. - *Cước lưu lượng vượt ngưỡng* tính theo công thức *Cước internet* của gói Basic.
Data Fix	Tương tự gói Basic + Giảm 10% giá cước	Mức cố định 1.000.000 đồng

Hãy vẽ sơ đồ lớp và viết chương trình cho phép công ty quản lý các hợp đồng trong một danh sách duy nhất với 2 chức năng: cho phép khách hàng đăng ký hợp đồng mới và thông báo tiền cước cho tất cả khách hàng vào cuối tháng.

Yêu cầu:

- Áp dụng kế thừa và đa hình để tăng tính tái sử dụng và tính dễ mở rộng của chương trình.
- Sơ đồ lớp: thể hiện các lớp đối tượng, quan hệ giữa các lớp, các thuộc tính, các hàm trong mỗi lớp.
- Mã nguồn: ngôn ngữ C++.

Lưu ý: Các đoạn mã nguồn câu 1 và 2 đã được #include đầy đủ các thư viện cần thiết.

Câu 1:

a) Hãy cho biết kết quả trên màn hình của đoạn mã nguồn bên dưới. Giải thích ngắn gọn.

```
class A {
public:
   virtual void f( int x ) { cout << x << " "; }
};
class B: public A {
public:
   void f( int y ) { A::f( y + 1 ); }
};
void doSomething( A a, B b ) { a.f( 3 ); b.f( 3 ); }
void main() {
   B x, y;
   doSomething( x, y );
}
```

b) Hãy cho biết đoạn mã nguồn bên dưới gặp vấn đề gì? Cài đặt cụ thể cách giải quyết.

```
class A {
public:
   A() { a = new int[3]; for ( int i = 0; i < 3; i++ ) a[i] = i + 1; }
  ~A() { delete[] a; }
private:
   int *a;
};
void init( A &a ) {
   A b;
   a = b;
}
void main() {
   A obj;
   init( obj );
}
```

c) Cho lớp đối tượng: `class Singleton { };`

Hãy đề xuất một giải pháp để lớp đối tượng Singleton chỉ được phép tạo một thể hiện duy nhất. Cài đặt cụ thể giải pháp và giải thích ngắn gọn những yếu tố nào đảm bảo tính duy nhất đó.

Câu 2:

Hãy cài đặt lớp `BigInteger` biểu diễn số nguyên lớn có những thuộc tính và phương thức cần thiết để đoạn mã nguồn ở trang sau biên dịch thành công và chạy đúng ngữ nghĩa:

```
void main() {
   BigInteger n1(" 1234567891011 ");  // Khởi tạo từ chuỗi.
   BigInteger n2;                     // Khởi tạo mặc định = 0.
   cout << "Input an integer = ";
   cin >> n2;                         // Nhập giá trị từ bàn phím.
```

```
   if ( n1 == n2 )                    // So sánh bằng nhau.
      cout << "Equal.\n";
   else
      cout << "Not equal.\n";
}
```

Câu 3:

Chợ OOP-Market là một khu kinh doanh sầm uất của thành phố. Mặt bằng chợ được chia thành các sạp có diện tích khác nhau cho tiểu thương thuê. Có ba loại sạp tương ứng với ba loại mặt hàng được phép kinh doanh tại chợ là: thực phẩm, quần áo, và trang sức. Đơn giá thuê ở mỗi loại sạp là như nhau: 40.000.000 đồng/m^2/năm. Thông tin chung để quản lý mỗi sạp bao gồm:

- Số thứ tự sạp.
- Diện tích sạp (m^2).

Vào cuối năm, số tiền mỗi tiểu thương thuê sạp phải đóng = tiền thuê sạp + thuế doanh thu.
Tiền thuê sạp = đơn giá thuê * diện tích sạp.
Thuế doanh thu tùy thuộc vào từng loại sạp và cho bởi bảng sau:

Loại sạp	**Thuế doanh thu**
Sạp thực phẩm	5%
Sạp quần áo	10%
Sạp trang sức	- Phần doanh thu < 100.000.000 đồng: 20% - Phần doanh thu >= 100.000.000 đồng: 30%

Ngoài ra, những tiểu thương thuê sạp thực phẩm phải đóng thêm tiền sử dụng dịch vụ đông lạnh để bảo quản thực phẩm mà sạp của mình đã sử dụng trong năm (số tiền khác nhau ở từng sạp).

Hãy vẽ sơ đồ lớp và cài đặt chương trình cho phép quản lý số tiền mỗi sạp được thuê phải đóng hàng năm như sau:

a) Nhập vào danh sách thông tin các sạp được thuê.
b) Tính tổng số tiền các sạp được thuê phải đóng hàng năm.

Yêu cầu:

- Áp dụng kế thừa và đa hình để tăng tính tái sử dụng và tính dễ mở rộng của chương trình.
- Sơ đồ lớp: thể hiện các lớp đối tượng, quan hệ giữa các lớp, các thuộc tính, phương thức trong mỗi lớp.
- Mã nguồn: ngôn ngữ C++.

Tên học phần: **Phương pháp Lập trình Hướng đối tượng**

Thời gian làm bài: **110 phút**

Câu 1 (*05 điểm*)

```cpp
#include <iostream>
#include <sstream>
#include <string>
#include <typeinfo>
using namespace std;
const double PI = 3.1415928;
class Shape
{
public:
  Shape(){ cerr<<"construct Shape"<<endl; }
  virtual string Description() {
    stringstream ss;
    ss<<"Shape(area=" << this->Area() << ")";
    return ss.str();
  }
  virtual float Area() = 0;
  virtual void Scale(float scaleFactor) = 0;
  const type_info& InterfaceType() {
    return typeid(Shape);
  }
  const type_info& ImplementationType() {
    return typeid(*this);
  }
  virtual ~Shape(){
    cerr<<"destruct Shape"<<endl;
  }
};

class Circle : public Shape
{
private:
  float radius;
public:
  Circle(float r) :Shape(), radius(r) {
    cerr << "construct Circle" << endl;
  }
  string Description() {
    stringstream ss;
    ss << "Circle(r=" << this->radius << ")";
    return ss.str();
  }
  float Area() { return PI*radius*radius; }
  void Scale(float scaleFactor) {
    this->radius *= scaleFactor;
  }
  const type_info& InterfaceType() {
    return typeid(Circle);
  }
  ~Circle(){ cerr<<"destruct Circle"<<endl; }
};
```

```cpp
class Ellipse : public Shape
{
private:
  float majorR, minorR;
public:
  Ellipse(float a, float b)
    :Shape(), majorR(a), minorR(b)
  {
    cerr << "construct Ellipse" << endl;
  }
  string Description() {
    stringstream ss;
    ss << "Ellipse(a=" << this->majorR
       << ", b=" << this->minorR << ")";
    return ss.str();
  }
  float Area(){ return PI*majorR*minorR; }
  void Scale(float scaleFactor) {
    this->majorR *= scaleFactor;
    this->minorR *= scaleFactor;
  }
  void Scale(float sa, float sb) {
    this->majorR *= sa;
    this->minorR *= sb;
  }
  const type_info& InterfaceType() {
    return typeid(Ellipse);
  }
  ~Ellipse(){
    cerr<<"destruct Ellipse"<<endl;
  }
};
```

```
83 int main()
84 {
85   Ellipse ellipse(2.0/3, 6);
86   Shape *shape = &ellipse;
87   shape->Scale(1.0/2);
88   //shape->Scale(3, 1.0/3);
89
90   cout << "ellipse: " << endl;
91   cout << ellipse.Description() << endl;
92   cout << ellipse.Area() << endl;
93   cout << ellipse.InterfaceType().name() << endl;
94   cout << ellipse.ImplementationType().name() << endl;
95
96   cout << "shape = &ellipse: " << endl;
97   cout << shape->Description() << endl;
98   cout << shape->Area() << endl;
99   cout << shape->InterfaceType().name() << endl;
100  cout << shape->ImplementationType().name() << endl;
101
102  //Shape ashape;
103  //cout << ashape.Description() << endl;
104
105  return 0; //no error
106}
```

Chú thích về chương trình C++ bên trên:

- `cerr` và `cout` là 2 luồng xuất chuẩn đều **xuất ra màn hình console**.
- `typeid` là toán tử tính kiểu của một biểu thức, trả về đối tượng `type_info`.
  VD: `class X; typeid(X).name()=="X";`
  `X *a; typeid(a).name()=="X*";`

Hãy trả lời những câu hỏi liên quan tới chương trình bên trên:

a) Hãy viết kết quả xuất ra màn hình sau khi chương trình chạy xong.

b) Trong chương trình có nhiều hàm có cùng tên, chúng có quan hệ gì với nhau (hay không có quan hệ gì hết)? Hãy giải thích quan hệ giữa 2 hàm trong những cặp trùng tên sau: (Nêu tên quan hệ nếu có, nêu ý nghĩa và ứng dụng của quan hệ đó.)

- `Shape::Description()` và `Circle::Description()`
- `Ellipse::Scale(float)` và `Ellipse::Scale(float, float)`
- `Shape::InterfaceType()` và `Circle::InterfaceType()`
- `Circle::InterfaceType()` và `Ellipse::InterfaceType()`

c) Nếu bỏ dấu comment (//) của các dòng 88, 102 và 103 ra thì chương trình gặp lỗi gì?

d) [*] Phải thay thế vào chỗ của các dòng bị comment (88, 102 và 103) những dòng code như thế nào để thực hiện được ý đồ của các dòng code đó:

- Ở dòng 88 muốn gọi phương thức `Ellipse::Scale(float, float)` thông qua con trỏ `shape` (chứ không phải thông qua biến đối tượng `ellipse`).
- Ở dòng 102 và 103 muốn gọi được phương thức `Shape::Description()`.

* *<u>Lưu ý</u>: Những câu đánh dấu sao [*] là không bắt buộc (tính điểm cộng).*

Câu 2 (*05 điểm*)

Bệnh viện Nhân dân 115 là một trong những bệnh viện lớn tại TP.HCM. Hiện bệnh viện đang có nhu cầu xây dựng hệ thống quản lý hoạt động khám chữa bệnh của các bệnh nhân. Mỗi bệnh nhân được cấp 1 mã số (MSBN) để tiện việc quản lý danh sách các bệnh nhân. Bệnh viện cho phép khám điều trị nội trú và ngoại trú. Nếu cùng 1 người có cả hoạt động khám ngoại trú lẫn điều trị nội trú thì sẽ được cấp 2 mã số khác nhau và được quản lý riêng như 2 người khác nhau. Viện phí của từng bệnh nhân được tính như sau:

Với bệnh nhân ngoại trú: Bệnh viện thu một mức phí nhất định cho mỗi lần khám tùy theo mỗi loại bệnh lý.

Với bệnh nhân nội trú: Tiền khám chữa bệnh = số ngày nằm viện * (phí khám bệnh mỗi ngày + đơn giá phòng).

Có 3 loại phòng cho bệnh nhân điều trị nội trú chọn lựa với đơn giá:

- Phòng loại C, đơn giá 600.000 đ/ngày.
- Phòng loại B, đơn giá 900.000 đ/ngày.
- Phòng loại A, đơn giá 1.400.000 đ/ngày.

Mỗi hoạt động khám chữa bệnh điều được ghi nhận lại dưới dạng nhật ký (log hoạt động) theo tháng. Tùy theo từng loại hoạt động, mỗi dòng log sẽ có các thông tin tương ứng như sau:

- BN ngoại trú khám bệnh: Ngày, MSBN, "KB", phí khám chữa bệnh.
- BN nội trú nhập viện: Ngày, MSBN, "NV", phí khám chữa bệnh mỗi ngày, loại phòng.
- BN nội trú xuất viện: Ngày, MSBN, "XV".
- Tổng kết viện phí (cuối cùng trong log): Ngày, -1, "TKVP".

Trong đó:

- Ngày là một số nguyên từ 1 tới 31 thể hiện ngày trong tháng.
- MSBN là một số tự nhiên, được đánh số liên tiếp từ 1.
- "KB", "NV", "XV", "TKVP" là các chuỗi ký tự thể hiện loại hoạt động.

Ví dụ một log hoạt động 8 ngày đầu tiên của tháng 12:
(Giả sử chưa có bệnh nhân nào trong danh sách.)

Nội dung tập tin log hoạt động	Giải thích ý nghĩa & Viện phí tới ngày hôm đó (VPTL)
1 001 KB 20000	new BN[MSBN=001] khám bệnh => VPTL = 20kđ
1 002 NV 100000 B	new BN[MSBN=002] nhập viện vào phòng loại B, mỗi ngày khám chữa bệnh hết 100kđ
6 003 KB 50000	new BN[MSBN=003] khám bệnh => VPTL = 50kđ
6 001 KB 30000	BN[MSBN=001] khám bệnh => VPTL = 20kđ + 30kđ = 50kđ
6 004 NV 300000 C	new BN[MSBN=004] nhập viện vào phòng loại C, mỗi ngày khám chữa bệnh hết 300kđ
7 004 XV	BN[MSBN=004] xuất viện => VPTL = ***1****(300k+600kđ) = 900kđ
8 -1 TKVP	Tổng kết viện phí: + BN[MSBN=001] => VPTL = 50kđ + BN[MSBN=002] đã nằm 7 ngày ở phòng loại B => VPTL = ***7****(100kđ+900kđ) = 7000kđ + BN[MSBN=003] => VPTL = 50kđ + BN[MSBN=004] => VPTL = 900kđ = => Tổng viện phí = 50kđ + 7000kđ + 50kđ + 900kđ = 8000kđ

a) Hãy áp dụng các tính chất hướng đối tượng đã học để vẽ sơ đồ lớp (cả chi tiết các thành phần trong lớp lẫn quan hệ giữa các lớp) cho chương trình thực hiện các việc sau:

- Đọc dữ liệu các bệnh nhân từ tập tin nhật ký hoạt động để lập danh sách bệnh nhân (thay cho việc nhập liệu từ bàn phím như thông thường).
- In ra màn hình bảng thống kê viện phí của từng bệnh nhân và tổng viện phí bệnh viện thu vào trong tháng.
- Cho biết bệnh viện có bao nhiêu bệnh nhân nội trú và bao nhiêu bệnh nhân ngoại trú.

b) Viết chương trình cài đặt cho thiết kế ở câu **a**. (Cài đặt ngay trong khai báo lớp, không cần tách riêng khai báo lớp với định nghĩa hàm. Được phép sử dụng bộ thư viện chuẩn STL: string, vector, v.v…)

c) [*] Để việc điều trị hiệu quả hơn, các bác sĩ cần biết được lịch sử điều trị của từng bệnh nhân. Hãy nâng cấp hệ thống để bổ sung thêm hoạt động Lập hồ sơ bệnh án cho bệnh nhân. Hồ sơ bệnh án của mỗi bệnh nhân sẽ ghi lại mọi hoạt động của bệnh nhân đó trong tháng để có thể tra cứu lại các hoạt động khi cần thiết. Bạn sẽ thay đổi hay bổ sung thêm cho các lớp đối tượng như thế nào? Hãy giải thích và minh họa thêm trên sơ đồ lớp ở câu **a**.

** <u>Lưu ý</u>: Những câu đánh dấu sao [*] là không bắt buộc (tính điểm cộng).*