СТРУКТУРА ОТЧЕТА ПО РАСЧЕТНОМУ ЗАДАНИЮ

ПО КУРСУ "ООП: Архитектурное проектирование и паттерны программирования"

[1 СТРУКТУРА ОТЧЕТА 1](#_Toc94628752)

[2 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ 1](#_Toc94628753)

[3 ПРИМЕР ОПИСАНИЯ ОБЪЕКТОВ И ФУНКЦИОНАЛЬНЫХ ВОЗМОЖНОСТЕЙ 2](#_Toc94628754)

[4 ОБЪЕКТЫ И КЛАССЫ ПРОЕКТИРУЕМОЙ СИСТЕМЫ 3](#_Toc94628755)

[5 ПРИМЕР РЕАЛИЗАЦИИ ПОДСИСТЕМЫ НА ОСНОВЕ ПРИНЦИПА ДЕЛЕГИРОВАНИЯ 4](#_Toc94628756)

[6 ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ 7](#_Toc94628757)

[7 ПРИМЕР РЕАЛИЗАЦИИ ПОДСИСТЕМЫ С ИСПОЛЬЗОВАНИЕМ PROXY 11](#_Toc94628758)

[8 ТЕКСТ ПРОГРАММЫ С ИСПОЛЬЗОВАНИЕМ PROXY 11](#_Toc94628759)

# 1 СТРУКТУРА ОТЧЕТА

1. Титульный лист
2. Задание
3. Описание предметной области
4. Oбъекты и классы проектируемой системы
5. Пример реализации подсистемы на основе принципа делегирования
6. Список источников

Далее будут приведены примеры реализации некоторых пунктов отчета на следующем примере:

Прикладная область для выполнения лабораторных работ: «Интерактивные логические игры и головоломки для детей детей дошкольного и младшего школьного возраста»,

Тема: Логическая задача «Волк, Коза, Капуста».

# 2 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

Описание предметной области содержит около 1 страницы текста с описанием:

- целей разработки соответствующих программ,

- решаемых задач,

- примеров существующих программ.

# 3 ПРИМЕР ОПИСАНИЯ ОБЪЕКТОВ И ФУНКЦИОНАЛЬНЫХ ВОЗМОЖНОСТЕЙ

Характеристика поставленной задачи

Лодочник должен перевести на лодке с одного берега на другой три объекта: волка, козу и капусту. В лодке, кроме самого лодочника, может поместиться только один перевозимый объект, поэтому лодочнику придется совершить несколько рейсов. Но, если он оставит без присмотра на одном берегу волка и козу, то волк съест козу. А если вместе на берегу окажутся коза и капуста, то коза съест капусту Задача: как лодочник должен перевезти в целости и сохранности все три объекта с одного берега на другой?

Перечень базовых объектов и их функционала

Лодка: транспортное средство, имеющее двигатель некоторого типа: умеет передвигаться между пунктами, запуская, а затем останавливая мотор. Лодка: может быть разных типов, иметь разные двигатели двигатель (от резиновой моторки до атомной подлодки, летающей амфибии и т.п.).

Мотор: используется для перемещения транспортного средства. Умеет запускаться и останавливаться. В качестве мотора в простейшем случае будем рассматривать бензиновый двигатель и весла. В общем случае тип двигателя ничем не ограничен.

Перевозимые объекты: пассажиры транспортного средства. Иногда требуют специальных условий для перевозки, так как могут обладать свойством опасности для других объектов (могут съесть другой объект, быть ядовитым, радиоактивным и т.п.).

Лодочник: водитель транспртного средства. Выбирает стратегию перевозки объектов, обеспечивает контроль безопасности объектов, дает команду объектам переместиться в лодку или из лодки, управляет лодкой. Лодочник может иметь множество стратегий для выбора перевозимых объектов.

Новые действующие объекты и возможное расширение системы

Баба Яга: следит за оставшимися без надзора объектами и планирует их похищение.

Ступа - транспорт Бабы Яги - умеет выполнять различные типы перемещения, при этом может маскироваться под лодку.

Для реализации первой итерации системы рассмотрим объекты в минимальном варианте, обеспечиваюш\щем решение задачи.

# 4 ОБЪЕКТЫ И КЛАССЫ ПРОЕКТИРУЕМОЙ СИСТЕМЫ

Перечень классов, интерфейсов и объектов

Класс "перевозимый объект" IPassenger

Конкретные объекты Волк, Коза, Капуста, ...

Класс "вредоносность" IDanger

Классы-наследники конкретных вредоносностей: Убивать, Съедать, Заражать, ...

Класс "набор вредоносностей" - IComposite

Наборы вредоносностей для конкретных объектов класса IPassenger

Класс "транспортное средство ITransport

Классы-наследники: Плавающие, Летающие, ...

Конкретные объекты: Лодка, Вертолет, Ступа, ...

Класс "мотор транспортного средства" - IEngine

Классы-нвследники: ручная тяга (весла), бензиновый двигатель, ...

Конкретные объекты: Весло, Мотор, Атомный двигатель. Атомный двигатель…..

Класс "водитель транспортного средства" IDriver

Конкретные объекты: Лодочник, Капитан, Баба Яга, ...

Класс "стратегия решения задачи" IStrategy

Классы-наследники: Простая стратегия № 1, ...

Класс "состояние игры IStatee

Классы-наследники конкретных состояний: состояние-1, состояние-2, ...

Класс "внешние силы - наблюдатели" IObserver

Конкретные объекты: Баба Яга, ...

Отношения между классами

ITransport --- композиция ----> IEngine

IDriver --- композиция ----> ITransport

IState --- агрегация ----> IStrategy

IDanger --- агрегация ----> IPassenger

IObserver --- агрегация ----> IPassenger

IComposite --- наследование ----> IDanger

IPassenger --- агрегация ----> IComposite

# 5 ПРИМЕР РЕАЛИЗАЦИИ ПОДСИСТЕМЫ НА ОСНОВЕ ПРИНЦИПА ДЕЛЕГИРОВАНИЯ

Для выполнения задания по теме "Делегирование" реализуем простейший вариант системы, оставим в системе только следующие три перевозимых объекта, при этом клиент (функция main) использует фиксированную стратегию, определенную в классе Solution, который задает порядок перевозимых объектов.

Конструктивные элементы:

1) IDriver делегирует ITransport

- действие "загрузить транспортное средство"

- действие "выполнить перевозку в пункт назначения"

2) ITransport делегирует IEngine

- действие "выполнить маршрут"

3) Введем в систему защитного заместителя Proxy для контроля прав водителя на управление транспортным средством. доступности и безопасности нового состояния типа IState в соответствии с

системой вредоносностей объектов, находящихся в одном месте.

4) Продемонстрируем работоспособность системы для различных используемых транспортных средств с различными моторами. С этой целю создадим три транспортных средства: Лодку с веслами, Лодку с бензиновым мотором, Вертолет.

Программа решения задачи не должна изменяться при замене одного транспортного средства на другое.

Код программы

class IPassenger{

private:

string myName;

public:

IPassenger(string s) { myName = s; }

void name() { cout << myName << endl; }

};

// интерфейсы/абстрактные классы делегатов

class IEngine{

private:

string myName;

public:

IEngine(string s){myName = s;}

virtual void actionStop() = 0; // интерфейс не имеет реализации

virtual void actionStart() = 0; // интерфейс не имеет реализации

void name() { cout << myName << endl; }

};

class ITransport{

private:

IEngine \* motor;

public:

ITransport (IEngine \* m) { motor = m; }

virtual void move() = 0; // интерфейс не имеет реализации

void setEngine (IEngine \* m) { motor = m; }

void actionStart() {motor -> actionStart();}

void actionStop() {motor -> actionStop();}

};

// классы конкретных делегатов

// конкретный класс "лодочные весла"

class BoatOars: public IEngine{

public:

BoatOars():IEngine("BoatOars") {}

virtual void actionStart() {cout << " lower the boat oars into the water" << endl;}

virtual void actionStop() {cout << " lift the boat oars out of the water" << endl;}

};

// конкретный класс "Бензиновый двигатель"

class GasEngine : public IEngine{

public:

GasEngine():IEngine("GasEngine"){}

virtual void actionStop() {cout << " GasEngine stop" << endl;}

virtual void actionStart() {cout << " GasEngine start" << endl;}

};

// ВЕРТОЛЕТ

class ClassHelicopter : public ITransport{

public:

virtual void move(){

actionStart();

cout << " I am flying! " << endl;

actionStop();

}

ClassHelicopter(IEngine \* m):ITransport(m) {}

};

//ЛОДКА

class ClassBoat : public ITransport{

public:

virtual void move(){

actionStart();

cout << " I am swiming! " << endl;

actionStop();

}

ClassBoat(IEngine \* m):ITransport(m) {}

};

// класс лодочник

class IDriver{

private:

ITransport \* transport;

IPassenger \* passanger;

public:

void setTransport(ITransport \* t) { transport = t; }

void setPassanger(IPassenger \* p) { passanger = p; }

void moveRight() {

cout << "Passanger " ; passanger -> name() ;

cout << " we are moving to right direction " << endl;

transport -> move () ;

}

void moveLeft() {

cout << "Passanger " ; passanger -> name() ;

cout << " we are moving to left direction " << endl;

transport -> move () ;

}

};

class Solution { // класс, решающий задачу

public:

void sol(IDriver \* h) {

IPassenger \* empty = new IPassenger ("Empty"),

\* cabbage = new IPassenger ("Cabbage"),

\* wolf = new IPassenger("Wolf"),

\* goat = new IPassenger("Goat");

h -> setPassanger(goat);

h -> moveRight();

h -> setPassanger(empty);

h -> moveLeft();

h -> setPassanger(wolf);

h -> moveRight();

h -> setPassanger(goat);

h -> moveLeft();

h -> setPassanger(cabbage);

h -> moveRight();

h -> setPassanger(empty);

h -> moveLeft();

h -> setPassanger(goat);

h -> moveRight();

delete(empty);

delete(cabbage);

delete(wolf);

delete(goat);

}

};

\// Клиентская программа

int main() {

// создаем объекты двигателей

IEngine \* d1 = new BoatOars(),

\* d2 = new GasEngine ();

// создаем объекты траспортных средств

ITransport \* h = new ClassHelicopter (d2),

\* b1 = new ClassBoat(d1),

\* b2 = new ClassBoat(d2);

// создаем объект лодочника

IDriver \* human = new IDriver();

Solution s;

cout << " BOAT WHITH BOAT OARS" << endl;

human -> setTransport(b1);

s.sol(human);

cout << endl;

cout << " BOAT WHITH ENGINE" << endl;

human -> setTransport(b2);

s.sol(human);

cout << endl;

cout << " HELICOPTER" << endl;

human -> setTransport(h);

s.sol(human);

cout << endl;

delete(d1);

delete(d2);

delete(h);

delete(b1);

delete(b2);

delete(human);

return 0;

}

# 6 ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

Использованные в программе объекты демонстрируют единую работу с разными транспортными средствами (лодка и вертолет), а также с разными способами приведения в движение транспортного средства (лодка на веслах и лодка с бензиновым мотором). Предложенная реализация позволяет легко расширять систему и вводить в нее новые транспортные средства с новыми моторами без изменения существующего кода.

Вывод результатов в процессе решения

BOAT WHITH BOAT OARS

Passanger Goat

we are moving to right direction

lower the boat oars into the water

I am swiming!

lift the boat oars out of the water

Passanger Empty

we are moving to left direction

lower the boat oars into the water

I am swiming!

lift the boat oars out of the water

Passanger Wolf

we are moving to right direction

lower the boat oars into the water

I am swiming!

lift the boat oars out of the water

Passanger Goat

we are moving to left direction

lower the boat oars into the water

I am swiming!

lift the boat oars out of the water

Passanger Cabbage

we are moving to right direction

lower the boat oars into the water

I am swiming!

lift the boat oars out of the water

Passanger Empty

we are moving to left direction

lower the boat oars into the water

I am swiming!

lift the boat oars out of the water

Passanger Goat

we are moving to right direction

lower the boat oars into the water

I am swiming!

lift the boat oars out of the water

BOAT WHITH ENGINE

Passanger Goat

we are moving to right direction

GasEngine start

I am swiming!

GasEngine stop

Passanger Empty

we are moving to left direction

GasEngine start

I am swiming!

GasEngine stop

Passanger Wolf

we are moving to right direction

GasEngine start

I am swiming!

GasEngine stop

Passanger Goat

we are moving to left direction

GasEngine start

I am swiming!

GasEngine stop

Passanger Cabbage

we are moving to right direction

GasEngine start

I am swiming!

GasEngine stop

Passanger Empty

we are moving to left direction

GasEngine start

I am swiming!

GasEngine stop

Passanger Goat

we are moving to right direction

GasEngine start

I am swiming!

GasEngine stop

HELICOPTER

Passanger Goat

we are moving to right direction

GasEngine start

I am flying!

GasEngine stop

Passanger Empty

we are moving to left direction

GasEngine start

I am flying!

GasEngine stop

Passanger Wolf

we are moving to right direction

GasEngine start

I am flying!

GasEngine stop

Passanger Goat

we are moving to left direction

GasEngine start

I am flying!

GasEngine stop

Passanger Cabbage

we are moving to right direction

GasEngine start

I am flying!

GasEngine stop

Passanger Empty

we are moving to left direction

GasEngine start

I am flying!

GasEngine stop

Passanger Goat

we are moving to right direction

GasEngine start

I am flying!

GasEngine stop

# 7 ПРИМЕР РЕАЛИЗАЦИИ ПОДСИСТЕМЫ С ИСПОЛЬЗОВАНИЕМ PROXY

Дополним реализованный вариант системы использованием паттерна Proxy (Заместителя). Выберем тип защитного заместителя ProxiDriver, который будет осуществлять доступ к транспортномк средству только такого водителя, у которого есть права на управление данным типом транспортного средства.

Введем тип прав на управление транспортным средством:

enum DriverLicense {EMPTYLICENSE, BOAT, HELICOPTER, AIRPLANE, CAR};

Установим информацию о правах в классе ITransport и в классе IDriver, а в классе заместителей ProxiDriver будем проверять соответствие прав. При неравенстве типов прав движение не выполняется. Дополним функцию main работой с ProxyDriver, который разрешает работу конкретного водителя при соответствии прав (управление лодкой с правами на управление лодкой) и не разрешает в противном случае (управление вертолетом с правами на лодку).

int main() {

…

// Работа с Proxy:

cout << endl << "Proxy witth last Human (BOAT) and Transport (HELICOPTER)" << endl;

IDriver \* proxy1 = new ProxyDriver(human);

// human пытается работать с вертолетом

s.sol(proxy1);

cout << endl << "Proxy after reset Human (BOAT) and Transport (BOAT)" << endl;

human -> setTransport(b1);

IDriver \* proxy2 = new ProxyDriver(human );

…

}

Получим вывод для работы с ProxyDriver. Каждпя попытка выполнить передвижение блокируется:

Proxy witth last Human (BOAT) and Transport (HELICOPTER)"

h -> setPassanger(goat)

start moveRight

License is absent!

…

License is absent!

License is absent!

# 8 ТЕКСТ ПРОГРАММЫ С ИСПОЛЬЗОВАНИЕМ PROXY

Приведем только изменившиеся классы. Все изменения связаны только с тем, что у транспортного средства и у водителя появилось поле типа DriverLicense, а также функции записи и чтения этого поля. Весь остальной код остается без изменений.

// Класс водителя транспортного средства ( в частности, лодочник )

// дополнен полем прав у водителя с функциями установки и получения типа прав

class IDriver{

private:

DriverLicense license; // имеющиеся права у водителя {BOAT, HELICOPTER, AIRPLANE, CAR};

ITransport \* transport;

IPassenger \* passanger;

public:

IDriver(DriverLicense lic) {license = lic;}

void setTransport(ITransport \* t) { transport = t; }

void setPassanger(IPassenger \* p) { passanger = p; }

virtual void moveRight() {

cout << "Passanger " ; passanger -> name() ;

cout << " we are moving to right direction " << endl;

transport -> move () ;

}

virtual void moveLeft() {

cout << "Passanger " ; passanger -> name() ;

cout << " we are moving to left direction " << endl;

transport -> move () ;

}

DriverLicense getLicense(){ return license; }

ITransport \* getTransport () { return transport; }

};

// Новый класс заместителя, контролирующего соответствие прав

class ProxyDriver : public IDriver{

private:

IDriver \* realDriver;

public:

ProxyDriver(IDriver \* driver): IDriver(EMPTYLICENSE) { realDriver = driver; }

virtual void moveRight() {

ITransport \* t = realDriver->getTransport();

if (t -> getLicense() == realDriver->getLicense() )

realDriver->moveRight();

else cout << "License is absent!" << endl;

}

virtual void moveLeft() {

ITransport \* t = realDriver->getTransport();

if (t -> getLicense() == realDriver->getLicense())

realDriver->moveRight();

else cout << "License is absent!" << endl;

}

};

Появилась информация о типе прав у класса транспортного средства:

class ITransport{

private:

DriverLicense license; // необходимые права у водтьеля

IEngine \* motor;

public:

ITransport (IEngine \* m, DriverLicense l) { motor = m; license = l; }

virtual void move() = 0; // интерфейс не имеет реализации

void setEngine (IEngine \* m) { motor = m; }

void actionStart() {motor -> actionStart();}

void actionStop() {motor -> actionStop();}

DriverLicense getLicense(){ return license;}

void setLicense(DriverLicense l){license = l;}

};