**Задания к дисциплине «Паттерное программирование»**

1. Разработать информационную систему для моделирования соревнования в беге между различными животными.

2. Разработать информационную систему для моделирования геобиоценоза.

3. Реализовать иерархию классов согласно следующей схеме:



4. Спроектировать и реализовать иерархию классов для игровых персонажей и разных типов вооружения. Каждый персонаж в любой момент времени использует только один вид оружия, но может свободно менять оружие в ходе игры.

5. Промоделировать чат на основе паттерна Observer.

6. Адаптировать лекционный пример из темы паттерн «Декоратор» в соответствии с новыми требованиями: Теперь кофе можно заказать в маленькой, средней или большой чашке. Cчитать размер порции неотъемлемой частью класса кофе, поэтому в класс Beverage добавить два новых метода: setSize() и getSize(). Стоимость дополнений также зависит от размера порции, так что, скажем, добавка сои должна стоить 10, 15 или 20 центов для маленькой, средней или большой порции соответственно.

#include <string>

#include <iostream>

using namespace std;

class Beverage

{

protected:

string description;

public:

virtual string getDescription() {return description;}

virtual double cost()=0;

};

class Espresso : public Beverage

{

public:

Espresso() {description="Espresso";}

virtual double cost() {return 1.99;}

};

class HouseBlend : public Beverage

{

public:

HouseBlend() {description="House Blend coffee";}

virtual double cost() {return 0.99;}

};

class DarkRoast : public Beverage

{

public:

DarkRoast() {description="Dark Roast coffee";}

virtual double cost() {return 1.39;}

};

class CondimentDecorator : public Beverage

{

protected:

Beverage \*beverage;

};

class Soy : public CondimentDecorator

{

public:

Soy(Beverage \*b) {beverage=b;}

virtual string getDescription() {return beverage->getDescription()+", Soy";}

virtual double cost() {return 0.20+beverage->cost();}

~Soy() {delete beverage;}

};

class Whip : public CondimentDecorator

{

public:

Whip(Beverage \*b) {beverage=b;}

virtual string getDescription() {return beverage->getDescription()+", Whip";}

virtual double cost() {return 0.15+beverage->cost();}

~Whip() {delete beverage;}

};

int \_tmain(int argc, \_TCHAR\* argv[])

{

Beverage \*b=new Espresso;

cout<<b->getDescription()<<", $"<<b->cost()<<endl;

delete b;

Beverage \*b2=new HouseBlend;

b2=new Soy(b2);

b2=new Whip(b2);

b2=new Whip(b2);

cout<<b2->getDescription()<<", $"<<b2->cost()<<endl;

delete b2;

return 0;

}

7. Доработать лекционный пример из темы паттерн «Команда»: Добавить устройство — трехскоростной вентилятор и реализовать функцию отмены последней операции.

#include <iostream>

#include <string>

using namespace std;

class Light

{

string descr;

public:

Light(string d) {descr=d;}

void on() {cout<<descr<<": Light is on"<<endl;}

void off() {cout<<descr<<": Light is off"<<endl;}

};

class GarageDoor

{

public:

void up() {cout<<"Garage door is open"<<endl;}

void down() {cout<<"Garage door is close"<<endl;}

};

class Stereo

{

public:

void on() {cout<<"Stereo is on"<<endl;}

void off() {cout<<"Stereo is off"<<endl;}

void setCd() {cout<<"Stereo is set for CD input"<<endl;}

void setDvd() {cout<<"Stereo is set for DVD input"<<endl;}

void setRadio() {cout<<"Stereo is set for radio"<<endl;}

void setVolume(int v) {cout<<"Stereo volume set to "<<v<<endl;}

};

class Command

{

public:

virtual void execute()=0;

};

class NoCommand : public Command

{

public:

virtual void execute() {}

};

class LightOnCommand : public Command

{

Light \*light;

public:

LightOnCommand(Light \*l) {light=l;}

virtual void execute() {light->on();}

};

class LightOffCommand : public Command

{

Light \*light;

public:

LightOffCommand(Light \*l) {light=l;}

virtual void execute() {light->off();}

};

class GarageDoorUpCommand : public Command

{

GarageDoor \*garageDoor;

public:

GarageDoorUpCommand(GarageDoor \*gd) {garageDoor=gd;}

virtual void execute() {garageDoor->up();}

};

class GarageDoorDownCommand : public Command

{

GarageDoor \*garageDoor;

public:

GarageDoorDownCommand(GarageDoor \*gd) {garageDoor=gd;}

virtual void execute() {garageDoor->down();}

};

class StereoOnWithCDCommand : public Command

{

Stereo \*stereo;

public:

StereoOnWithCDCommand(Stereo \*s) {stereo=s;}

virtual void execute()

{

stereo->on();

stereo->setCd();

stereo->setVolume(11);

}

};

class StereoOffCommand : public Command

{

Stereo \*stereo;

public:

StereoOffCommand(Stereo \*s) {stereo=s;}

virtual void execute() {stereo->off();}

};

//Пульт

class RemoteControl

{

int countButtons;

Command \*\*onCommands;

Command \*\*offCommands;

NoCommand \*noCommand;

public:

RemoteControl(int c)

{

countButtons=c;

onCommands=new Command\*[c];

offCommands=new Command\*[c];

noCommand=new NoCommand;

for(int i=0; i<c; i++)

{

onCommands[i]=noCommand;

offCommands[i]=noCommand;

}

}

~RemoteControl()

{

delete []onCommands;

delete []offCommands;

delete noCommand;

}

void setCommand(int slot, Command \*onCommand, Command \*offCommand)

{

if (onCommands[slot]!=noCommand) delete onCommands[slot];

onCommands[slot]=onCommand;

if (offCommands[slot]!=noCommand) delete offCommands[slot];

offCommands[slot]=offCommand;

}

void onButtonWasPushed(int slot) {onCommands[slot]->execute();}

void offButtonWasPushed(int slot) {offCommands[slot]->execute();}

};

int \_tmain(int argc, \_TCHAR\* argv[])

{

RemoteControl rc(7);

Light \*livingRoomLight = new Light("Living Room");

Light \*kitchenLight = new Light("Kitchen");

GarageDoor \*garageDoor = new GarageDoor;

Stereo \*stereo = new Stereo;

LightOnCommand \*livingRoomLightOn = new LightOnCommand(livingRoomLight);

LightOffCommand \*livingRoomLightOff = new LightOffCommand(livingRoomLight);

LightOnCommand \*kitchenLightOn = new LightOnCommand(kitchenLight);

LightOffCommand \*kitchenLightOff = new LightOffCommand(kitchenLight);

GarageDoorUpCommand \*garageDoorUp = new GarageDoorUpCommand(garageDoor);

GarageDoorDownCommand \*garageDoorDown = new GarageDoorDownCommand(garageDoor);

StereoOnWithCDCommand \*stereoOnWithCD = new StereoOnWithCDCommand(stereo);

StereoOffCommand \*stereoOff = new StereoOffCommand(stereo);

rc.setCommand(0, livingRoomLightOn, livingRoomLightOff);

rc.setCommand(1, kitchenLightOn, kitchenLightOff);

rc.setCommand(2, garageDoorUp, garageDoorDown);

rc.setCommand(3, stereoOnWithCD, stereoOff);

for(int i=0; i<7; i++)

{

rc.onButtonWasPushed(i);

rc.offButtonWasPushed(i);

}

return 0;

}