Федеральное агентство связи (Россвязь)

**Сибирский государственный университет**

**телекоммуникаций и информатики**

**КАФЕДРА ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ**

ДИСЦИПЛИНА

**Объектно-ориентированное программирование**

Лабораторная работа №9

Вариант - 4

Выполнил ст. гр.

ИУ-323

Токарь Т. М.

Проверил:

Доцент кандидат физико-математических наук Пудов С. Г.

Новосибирск 2014

Оглавление

[Паттерн Prototype: 3](#_Toc406202230)

[Паттерн Adapter: 7](#_Toc406202231)

**Задание:**

Реализовать два паттерна. Один по журналу и один на выбор.

**По журналу:** Prototype;

**На выбор:** Adapter.

# Паттерн Prototype:

**Тип паттерна:** Порождающий

**Паттерн Prototype (прототип) можно использовать в следующих случаях:**

* Система должна оставаться независимой как от процесса создания новых объектов, так и от типов порождаемых объектов. Непосредственное использование выражения new в коде приложения считается нежелательным.
* Необходимо создавать объекты, точные классы которых становятся известными уже на стадии выполнения программы.

**Описание паттерна Prototype:**

Для создания новых объектов паттерн Prototype использует прототипы. Прототип - это уже существующий в системе объект, который поддерживает операцию клонирования, то есть умеет создавать копию самого себя. Таким образом, для создания объекта некоторого класса достаточно выполнить операцию clone() соответствующего прототипа.

**Prototype с помощью выделенного класса-фабрики:**

**Файл «base.h»**

#ifndef BASE\_H

#define BASE\_H

**class Warrior**{

public:

virtual Warrior\* clone() = 0;

virtual void info() = 0;

virtual ~Warrior();

};

#endif // BASE\_H

**Файл «base.cpp»**

#include <base.h>

**Warrior::~Warrior()**{

}

**Файл «derivatives.h»**

#ifndef DERIVATIVES\_H

#define DERIVATIVES\_H

#include <base.h>

**class Infantryman: public Warrior**{

private:

friend class PrototypeFactory;

Infantryman();

public:

Warrior\* clone();

void info();

};

**class Archer: public Warrior**{

private:

friend class PrototypeFactory;

Archer();

public:

Warrior\* clone();

void info();

};

**class Horseman: public Warrior**{

private:

friend class PrototypeFactory;

Horseman();

public:

Warrior\* clone();

void info();

};

#endif // DERIVATIVES\_H

**Файл « derivatives.cpp»**

#include <derivatives.h>

#include <iostream>

using std::cout;

using std::endl;

//Infantryman - пехотинец

**Warrior\* Infantryman::clone()**{

return new Infantryman( \*this);

}

**void Infantryman::info(**){

cout << "Infantryman" << endl;

}

**Infantryman::Infantryman()**{

}

//Archer - стрелок

**Warrior\* Archer::clone()**{

return new Archer(\*this);

}

**void Archer::info()**{

cout << "Archer" << endl;

}

**Archer::Archer()**{

}

//Horseman - всадник

**Warrior\* Horseman::clone()**{

return new Horseman(\*this);

}

**void Horseman::info()**{

cout << "Horsmen" << endl;

}

**Horseman::Horseman()**{

}

**Файл «factory.h»**

#ifndef FACTORY\_H

#define FACTORY\_H

#include <base.h>

**class PrototypeFactory**{

public:

Warrior\* createInfantrman();

Warrior\* createArcher();

Warrior\* createHorseman();

};

#endif // FACTORY\_H

**Файл «factory.cpp»**

#include <factory.h>

#include <derivatives.h>

**Warrior\* PrototypeFactory::createInfantrman()**{

static Infantryman prototype;

return prototype.clone();

}

**Warrior\* PrototypeFactory::createArcher()**{

static Archer prototype;

return prototype.clone();

}

**Warrior\* PrototypeFactory::createHorseman()**{

static Horseman prototype;

return prototype.clone();

}

**Файл «main.cpp»**

#include <iostream>

#include <vector>

#include <factory.h>

using namespace std;

**int main()**{

PrototypeFactory factory;

vector<Warrior\*> v;

v.push\_back(factory.createInfantrman());

v.push\_back(factory.createArcher());

v.push\_back(factory.createHorseman());

for(size\_t i = 0; i < v.size(); ++i){

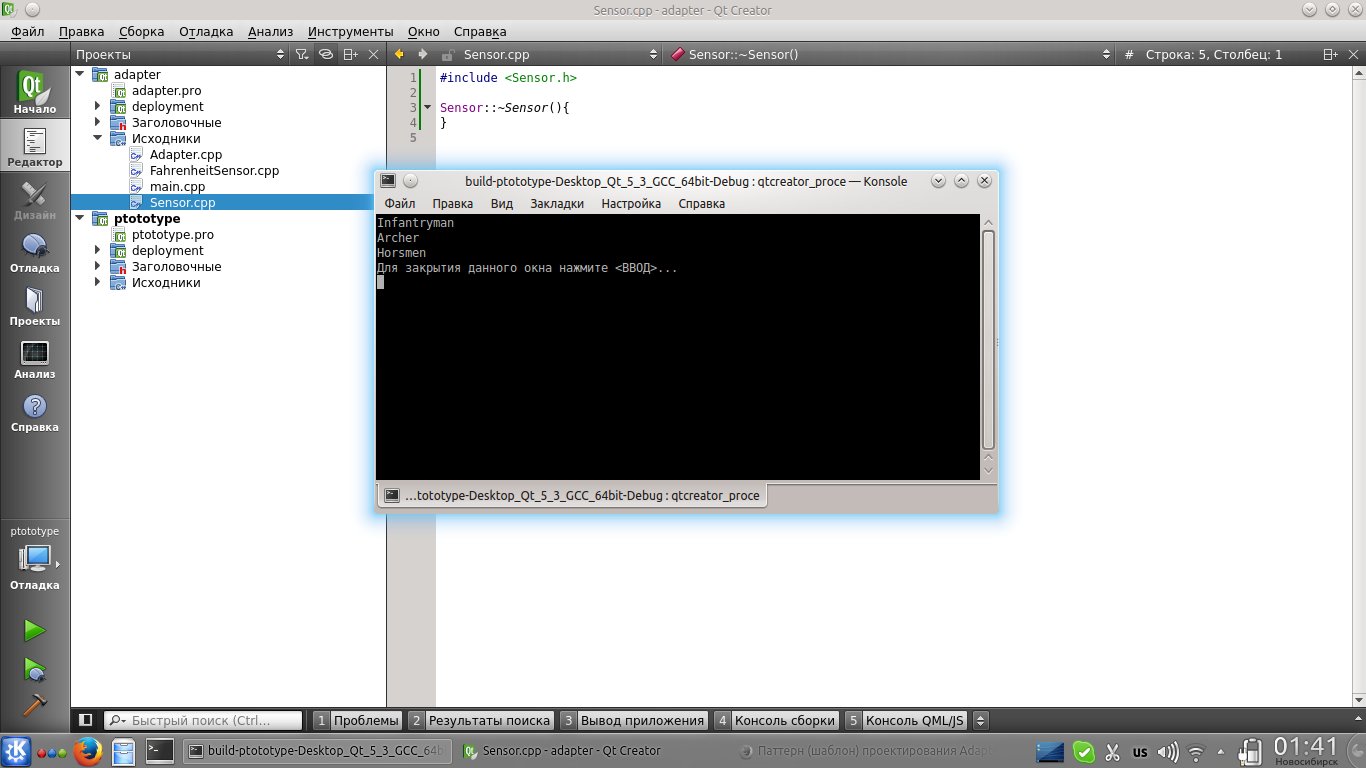
v[i]->info();

}

return 0;

}

**Результат работы:**



# Паттерн Adapter:

**Тип паттерна:** Структурный

**Описание паттерна Adapter:**

Пусть класс, интерфейс которого нужно адаптировать к нужному виду, имеет имя Adaptee. Для решения задачи преобразования его интерфейса паттерн Adapter вводит следующую иерархию классов:

* Виртуальный базовый класс Target. Здесь объявляется пользовательский интерфейс подходящего вида. Только этот интерфейс доступен для пользователя.
* Производный класс Adapter, реализующий интерфейс Target. В этом классе также имеется указатель или ссылка на экземпляр Adaptee. Паттерн Adapter использует этот указатель для перенаправления клиентских вызовов в Adaptee. Так как интерфейсы Adaptee и Target несовместимы между собой, то эти вызовы обычно требуют преобразования.

**Классическая реализация паттерна Adapter:**

**Файл «FahrenheitSensor.h»:**

#ifndef FAHRENHEITSENSOR\_H

#define FAHRENHEITSENSOR\_H

**class FahrenheitSensor**{

public:

float getFahrenheitTemp();

};

#endif // FAHRENHEITSENSOR\_H

**Файл «FahrenheitSensor.cpp»:**

#include <FahrenheitSensor.h>

**float FahrenheitSensor::getFahrenheitTemp()**{

float t = 33.0;

return t;

}

**Файл «Sensor.h»:**

#ifndef SENSOR\_H

#define SENSOR\_H

**class Sensor**{

public:

virtual ~Sensor();

virtual float getTemperature() = 0;

};

#endif // SENSOR\_H

**Файл «Sensor.cpp»:**

#include <Sensor.h>

**Sensor::~Sensor()**{

}

**Файл «Adapter.h»:**

#ifndef ADAPTER\_H

#define ADAPTER\_H

#include <Sensor.h>

#include <FahrenheitSensor.h>

**class Adapter: public Sensor**{

public:

Adapter(FahrenheitSensor\* p);

~Adapter();

float getTemperature();

private:

FahrenheitSensor\* p\_fsensor;

};

#endif // ADAPTER\_H

**Файл «Adapter.cpp»:**

#include <Adapter.h>

**Adapter::Adapter(FahrenheitSensor \*p): p\_fsensor(p)**{

}

**Adapter::~Adapter()**{

delete p\_fsensor;

}

**float Adapter::getTemperature()**{

return (p\_fsensor->getFahrenheitTemp()-32.0)\*5.0/9.0;

}

**Файл «main.cpp»:**

#include <iostream>

#include <Sensor.h>

#include <Adapter.h>

using namespace std;

int main()

{

Sensor\* p = new Adapter(new FahrenheitSensor);

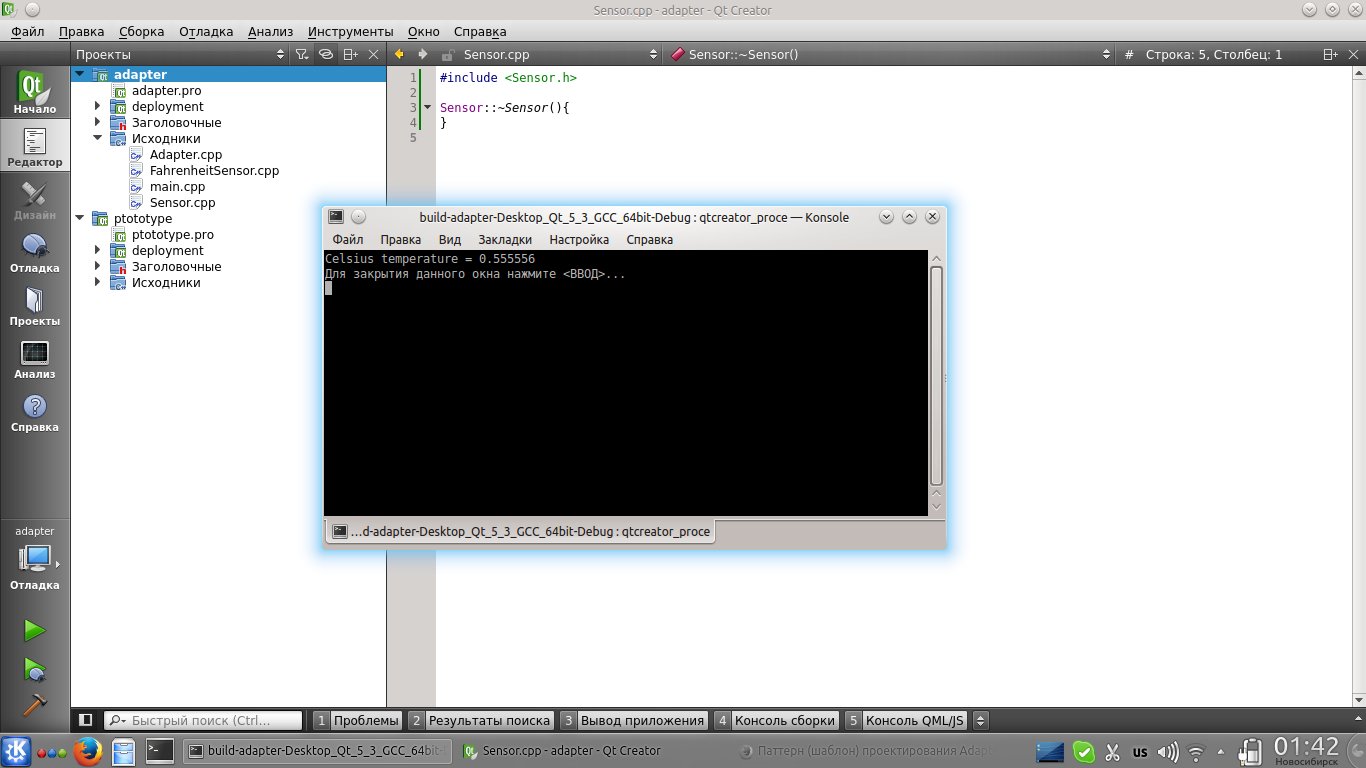
cout << "Celsius temperature = " << p->getTemperature() << endl;

delete p;

return 0;

}

**Результат работы:**

****