15.10.22

# **Конструирование объектов класса**

#include <string>

#include <vector>

#include <iostream>

using namespace std;

class Logger {

public:

Logger(const string& s) : nameOfClass(s) {

cout << nameOfClass << " birth " << endl;

}

~Logger() {

cout << nameOfClass << " Dead like a hero " << endl;

}

string nameOfClass;

};

class MultHeroes {

public:

MultHeroes(const string& s) : L(s + "MultHeroes") {}

string typeOfHero = "MultHeroes";

const Logger L;// = Logger("MultHeroes");

};

class Ogr : public MultHeroes {

public:

Ogr(const string& s) : MultHeroes(s), L(s) {

//typeOfHero = "MultHeroes";

}

Logger L = Logger("Ogr");

};

class FamilyOgr {

public:

FamilyOgr() : Shreck("Shreck"), Fiona("Fiona"), miniShreck("miniShreck") {}

Logger L = Logger("Family");

Ogr Shreck;

Ogr Fiona;

Ogr miniShreck;

};

int main() {

FamilyOgr family;

}

22.10.22

SOLID

S – single responsibility

(принцип единственной ответственности)

O – open-close

(принцип открытости-закрытости)

L – Liskov substitution

(принцип подстановки Барбары Лисков)

I – interface segregation

(принцип разделения интерфейсов)

D – dependency inversion

(принцип инверсии зависимостей)

Singleton – порождающий шаблон проектирования, гарантирующий, что в однопоточном приложении будет единственный экземпляр некоторого класса, и представляющий глобальную точку доступа к этому экземпляру.

Безопасность:

Проблемы с безопасностью singleton:

1. Рефлексия – изменение во время исполнения класса
2. Десериализация – возможность создать объект
3. Поток и небезопасность -
4. Проблемы с тестированием

29.10.22

**Как работает программа?**

#Multithreading – мультипоточность

Потоки:

* Single-threading – однопоточность(последовательное выполнение), следующая задача не выполнится, пока предыдущая не завершится.
* Hyper – threading – разделение задач на подзадачи для ускорения и оптимизации (не многопоточность).

Hyper – threading - это технология разработанная компанией Intel для повышения производительности процессоров собственного производства.

Принцип действия Hyper-Threading основывается на том, что в каждый момент времени только часть ресурсов процессора используется при выполнении программного кода. Неиспользуемые ресурсы также можно загрузить работой — например, задействовать для параллельного выполнения еще одного приложения (либо другого потока этого же приложения). В одном физическом процессоре Intel Xeon формируются два логических процессора (LP — Logical Processor), которые разделяют между собой вычислительные ресурсы CPU. Операционная система и приложения "видят" именно два CPU и могут распределять работу между ними, как и в случае полноценной двухпроцессорной системы.

Физическое ядро не будет бездействовать, а передаст управление потоку команд второго логического ядра. Таким образом, пока одно логическое ядро ожидает, например, данные из памяти, вычислительные ресурсы физического ядра будут использованы вторым логическим ядром.

* Multi – threading – многопоточность.

Multi – threading – свойство платформы или приложения, состоящее в том, что процесс, порождённый в операционной системе, может состоять из нескольких потоков, выполняющихся «параллельно», то есть без предписанного порядка во времени. При выполнении некоторых задач такое разделение может достичь более эффективного использования ресурсов вычислительной машины.

**Многопоточность** (как доктрину программирования) не следует путать ни с многозадачностью, ни с многопроцессорностью несмотря на то, что операционные системы, реализующие многозадачность, как правило, реализуют и многопоточность.

Пример:

#include <thread>

void function1()

{

cout << this\_thread::get\_id() << endl;

}

thread th1(function1);

th1.detach();//первый вариант завершения потока – завершение нового потока, при завершении основного.

//main thread

th1.join();// второй вариант завершения потока – ожидает завершения thread1.

Thread th1 (function1);

Thread th1 (function1, parameter1, parameter2);

Thread th1 (function1, std::ref(parameter1));//ref – класс ссылок

Thread th1 ([&result](){result = function1(parameter1,patameter2);});

«4 потока -одна консоль» - mutex.

5.11.22

«4 потока -одна консоль» - mutex.

Mutex – приметив синхронизации., обеспечивает взаимное исключение критических участков кода.

#include < thread>

Mutex mtx;

Mtx.lock();

Function1();

Mtx.unlock();

#include < thread>

Mutex mtx;

Void function1() //одновременно выполняется ток один поток, пока один поток не завершит выполнение, второй не начнет работать

{

Mtx.lock();

cout << this\_thread::get\_id() << endl;

mtx.unlock();

};

thread th1(function1);

th1.detach();

thread th2(function1);

th2.detach();

// main thread

#include < thread>

Mutex mtx;

Void function1()

{

Lock\_ guard<mutex>guard(mtx);//объяв без закрытия, закрывается при завершении функции

cout << this\_thread::get\_id() << endl;

};

thread th1(function1);

th1.detach();

thread th2(function1);

th2.detach();

// main thread

Recursive\_mutex rmtx;

Rmtx.lock();

Rmtx.unlock();

Uniqu\_lock<mutex> ul(mtx);

Unique\_lock<mutex> ul(mtx, std::defer\_lock);

Ul.lock();

Ul.unlock();