

ПМО ИУС

Конструкторы и деструкторы.  
Инкапсуляция.

Лекция 4

# Инициализация объектов

```
#include <iostream>

class time_t {
public:
    int m_hour;
    int m_min;
    int m_sec;
};

int main(int argc, char* argv[])
{
    time_t time;

    std::cout <<
        time.m_hour << ":" <<
        time.m_min << ":" <<
        time.m_sec <<
        std::endl;

    return 0;
}
```

1:256:1

# Конструкторы

```
class time_t {
public:
    int m_hour;
    int m_min;
    int m_sec;

    //! Устанавливает 00:00:00.
    time_t() {
        m_hour = 0;
        m_min = 0;
        m_sec = 0;
    }

    void
    display() {
        std::cout <<
            m_hour << ":" <<
            m_min << ":" <<
            m_sec <<
            std::endl;
    }
};

int main()
{
    time_t time;

    time.display();

    return 0;
}

0:0:0
```

# Конструкторы с параметрами

```
class time_t {  
[...]  
  
    ///  
    Устанавливает 00:00:00.  
    time_t() {  
        m_hour = 0;  
        m_min = 0;  
        m_sec = 0;  
    }  
  
    ///  
    Устанавливает время hour:min:sec.  
    time_t( int hour, int min, int sec ) {  
        m_hour = hour;  
        m_min = min;  
        m_sec = sec;  
    }  
  
    ///  
    Устанавливает время hour:00:00.  
    time_t( int hour ) {  
        m_hour = hour;  
        m_min = 0;  
        m_sec = 0;  
    }  
};
```

```
int main() {  
  
    time_t midnight;  
    time_t morning(8);  
    time_t afternoon(14, 10, 25);  
  
    midnight.display();  
    morning.display();  
    afternoon.display();  
  
    return 0;  
}  
  
0:0:0  
8:0:0  
14:10:25
```



# Конструкторы полей класса

```
class time_t {  
[...]  
  
    ///  
    *! Устанавливает время hour:min:sec.  
    *!  
    \param hour часы, 0..23.  
    \param min минуты, 0..59.  
    \param sec секунды, 0..59.  
  
    \note проверяет на корректность и  
    приводит к крайним значениям.  
    */  
    time_t( int hour, int min, int sec ) :  
        m_hour( hour ), m_min( min ), m_sec( sec ) {  
  
        if ( hour > 23 )  
            m_hour = 23;  
        else if ( hour < 0 )  
            m_hour = 0;  
  
        if ( min > 59 )  
            m_min = 59;  
        else if ( min < 0 )  
            m_min = 0;  
  
        if ( sec > 59 )  
            m_sec = 59;  
        else if ( sec < 0 )  
            m_sec = 0;  
  
    }  
};
```

```
int main()  
{  
    time_t time(20, 10, -1);  
  
    time.display();  
  
    return 0;  
}
```

20:10:0

# Конструкторы полей класса

```
#!/ Класс произошедшего события.
class event_t {
public:
    /*! Время, когда событие произошло.
    time_t m_time;

    /*! Описание произошедшего.
    std::string m_description;

    /*! Зафиксировать текущее время.
    event_t(
        /*! Время в часах.
        int hour,
        /*! Минуты.
        int min,
        /*! Секунды.
        int sec,
        /*! Описание события.
        const std::string & description ) :
        m_time( hour, min, sec ), m_description( description )
    {
        std::cout <<
            "Event '" << description << "' at " <<
            time.display() <<
            std::endl;
    }
};

int main()
{
    event_t event(10, 20, 25, "Start");

    return 0;
}

Event 'Start' at 10:20:25
```

# Ссылка на себя (this)

```
class time_t {
public:
    int hour;
    int min;
    int sec;

    //! Устанавливает время hour:00:00.
    time_t( int hour ) : min(0), sec(0) {

        if ( hour > 23 )
            this->hour = 23;
        else if (hour < 0)
            this->hour = 0;
        else this->hour = hour;
    }

    //! Конструктор копирования.
    /*!
        \param time объект копирования.
    */
    time_t( const time_t & time ) {
        (*this) = time;
    }
};
```

```
int main()
{
    time_t time(10);
    time_t time2(time);
    time2.display();

    return 0;
}

10:0:0
```

# Деструкторы

```
class simple_t {
public:
    std::string m_name;

    simple_t( const std::string & name )
    : m_name(name) {
        std::cout <<
            "Simple '" <<
            m_name.c_str() <<
            "' created." <<
            std::endl;
    }

    ~simple_t() {
        std::cout <<
            "Simple '" <<
            m_name.c_str() <<
            "' destroyed." <<
            std::endl;
    };
};
```

```
simple_t a( "Global" );

int main()
{
    std::cout <<
        "Begin" <<
        std::endl;

    simple_t b( "Local" );

    std::cout <<
        "End" <<
        std::endl;

    return 0;
}

Simple 'Global' created.
Begin
Simple 'Local' created.
End
Simple 'Local' destroyed.
Simple 'Global' destroyed.
```

# Захват и освобождение ресурсов

```
#!/ Динамический массив
#!/ задаваемой пользователем длины
#!/ с элементами типа int.
class dynamic_array_t {
public:
    /*! Указатель на начало массива.
    int * m_head;

    /*! Авто-создание массива
    /*! заданной длины.
    dynamic_array_t( int n ) {
        // При создании объекта
        // выделяется память.
        m_head = new int[n];
    }

    /*! Очищаем использованную память.
    ~dynamic_array_t() {
        // При уничтожении объекта
        // память освобождается.
        delete [] m_head;
    }
};

int main()
{
    int number;
    std::cin >> number;

    dynamic_array_t m(number);

    for( int i = 0; i < number; ++i )
        *(m.m_head+i) = i*i;

    for( int i = 0; i < number; ++i )
        std::cout << *(m.m_head+i) << " ";

    std::cout << std::endl;

    return 0;
}

10
0 1 4 9 16 25 36 49 64 81
```

# Создание функций по умолчанию

```
class time_t {  
[...]  
  
    // Пустой базовый конструктор.  
    // time_t() { }  
  
    // Пустой деструктор.  
    // ~time_t() { }  
  
    // Конструктор копирования.  
    // time_t( const time_t & time ) {  
    //     (*this) = time;  
    // }  
  
    // Оператор присваивания.  
    // time_t &  
    // operator = (const time_t & time ) {  
    //     (*this) = time;  
    // }  
};
```

```
int main()  
{  
    time_t time;  
  
    time.m_hour = 5;  
    time.m_min = 30;  
    time.m_sec = 20;  
  
    time_t time2 = time;  
    time_t time3(time);  
  
    time.display();  
    time2.display();  
    time3.display();  
  
    return 0;  
}
```

5:30:20

5:30:20

5:30:20

# Константные переменные

```
int main() {  
  
    // Задаем максимальный предел числа элементов.  
    const int c_max_number = 100;  
    // Горизонтальная координата.  
    int x[c_max_number];  
    // Вертикальная координата.  
    int y[c_max_number];  
  
    // Ввод необходимого числа элементов.  
    while(true) {  
        std::cin >> n;  
  
        if (( n <= c_max_number ) && ( n >= 0) )  
            break;  
  
        std::cout <<  
            "Wrong number! Must be 0.." << c_max_number << std::endl;  
    }  
  
    [...]  
}
```

# Константные ссылки и указатели

```
// Реальная переменная x.
```

```
int x = 10;
```

```
// Та же переменная x, но её нельзя изменить.
```

```
const int & y = x;
```

```
y = 5; //< Ошибка.
```

```
// Указатель только для чтения.
```

```
const int * p1 = x;
```

```
(*p1) = 1;
```

```
// Неизменяемый указатель на int.
```

```
int * const p2 = x;
```

```
p2 = p1;
```

```
// Неизменяемый указатель на int только для чтения.
```

```
const int * const p3 = x;
```

```
(*p3) = 3;
```

```
p3 = &x;
```

# Константные функции класса и константные объекты

```
class distance_t {
public:
    ///! Значение в метрах.
    float m_value;

    ///! Печать расстояния.
    void
    display() const {
        std::cout <<
            "distance:" << km_value() <<
            " km." <<
            std::endl;
    }

    ///! Установить новое расстояние (в метрах).
    void
    set_value( int value ) {
        m_value = value;
    }

    ///! Получить расстояние ( в км ).
    float
    km_value() const {
        return m_value / 1000;
    }
};
```

```
int main() {
    distance_t distance;
    distance.set_value( 10 );

    distance.display();

    const distance_t d( distance );
    ///! d.set_value(1);
    d.display();
}

0.01
0.01
```

# Инкапсуляция

Разработчик класса

Пользователь класса



# Инкапсуляция в C++ : интерфейс

```
class time_t {
public:

    //! Устанавливает 00:00:00.
    time_t();

    //! Устанавливает время hour:00:00.
    /*!
        \param hour часы, 0..23.
    */
    time_t( int hour );

    //! Устанавливает время hour:min:sec.
    /*!
        \param hour часы, 0..23.
        \param min минуты, 0..59.
        \param sec секунды, 0..59.

        \note проверяет на корректность и
              приводит к крайним значениям.
    */
    time_t( int hour, int min, int sec );

    //! Печать состояния объекта.
    void
    display();

private:
    int m_hour;
    int m_min;
    int m_sec;
};
```

# Инкапсуляция в C++ : реализация

```
time_t::time_t() :
m_hour( 0 ), m_min( 0 ), m_sec( 0 ) {
}

time_t::time_t( int hour, int min, int sec ) :
m_hour( hour ), m_min( min ), m_sec( sec ) {

    if ( hour > 23 )
        m_hour = 23;
    else if (hour < 0)
        m_hour = 0;

    if ( min > 59 )
        m_min = 59;
    else if (min < 0)
        m_min = 0;

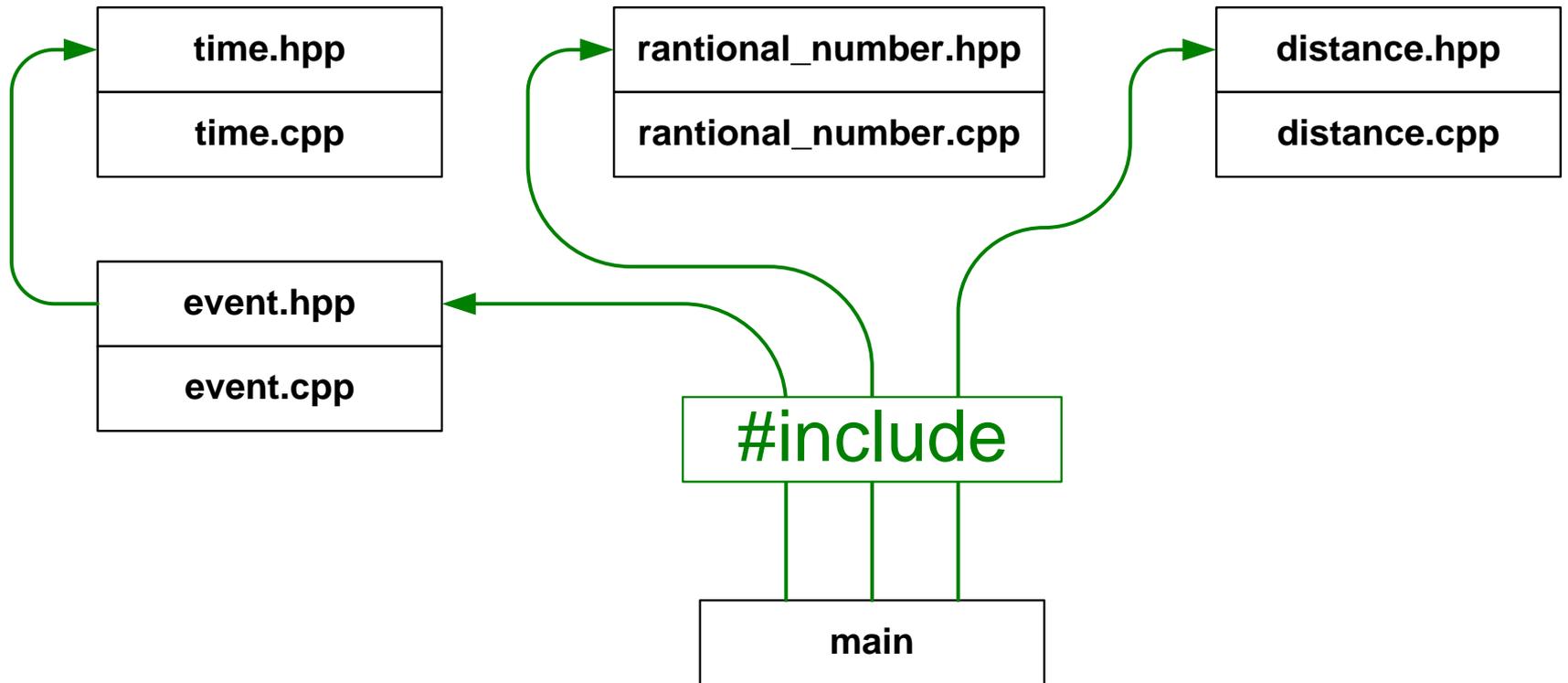
    if ( sec > 59 )
        m_sec = 59;
    else if (sec < 0)
        m_sec = 0;
}
```

```
time_t::time_t( int hour ) :
m_min(0), m_sec(0), m_hour( hour ) {

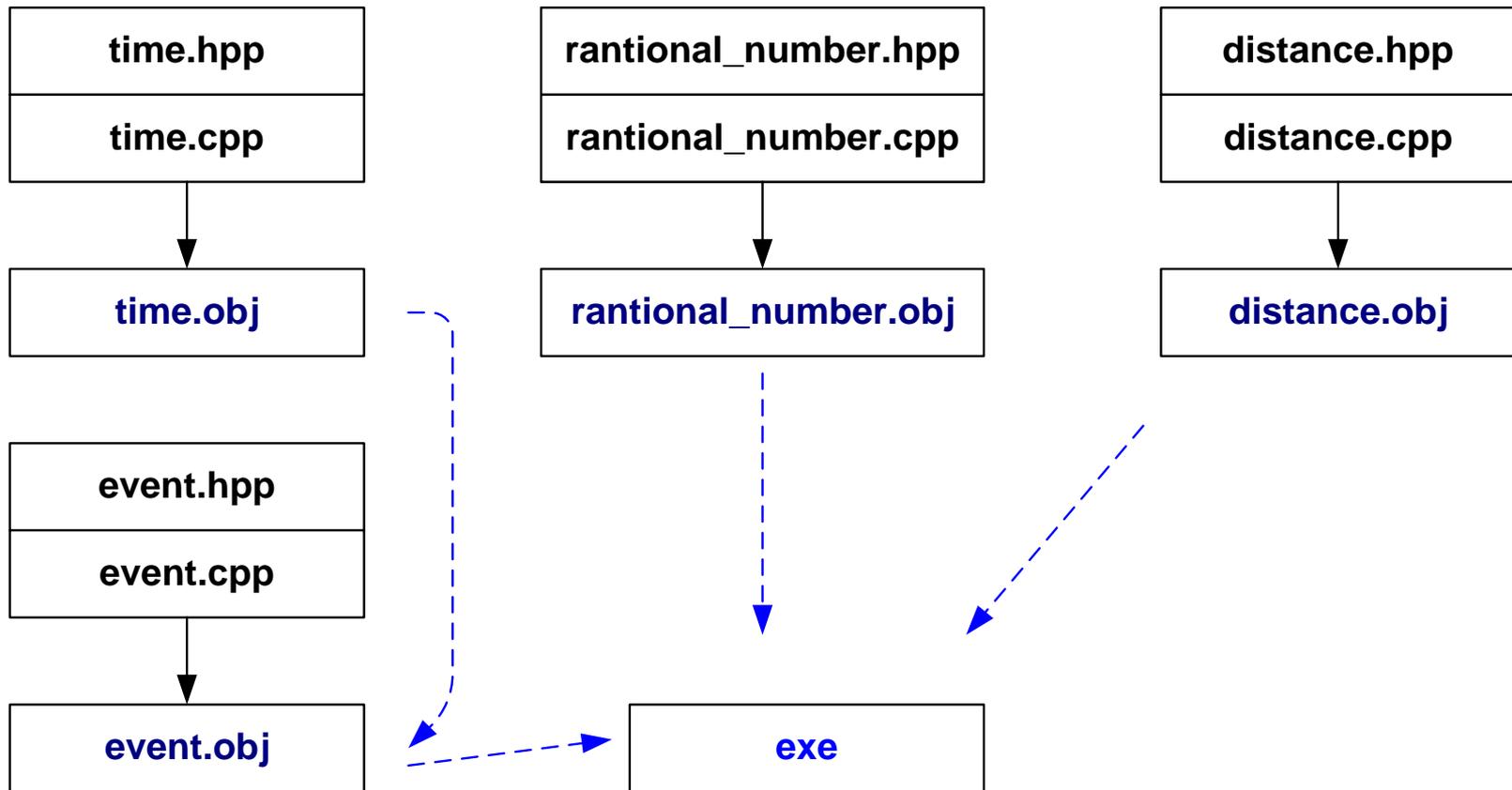
    if ( hour > 23 )
        m_hour = 23;
    else if (hour < 0)
        m_hour = 0;
    else m_hour = hour;
}

void
time_t::display() {
    std::cout <<
        m_hour << ":" <<
        m_min << ":" <<
        m_sec <<
        std::endl;
}
```

# Состав проекта



# Сборка проекта (Linking)



# Защита от повторного подключения

```
#if !defined( _EVENT_HPP_ )
```

```
#define _EVENT_HPP_
```

```
#include <time.hpp>
```

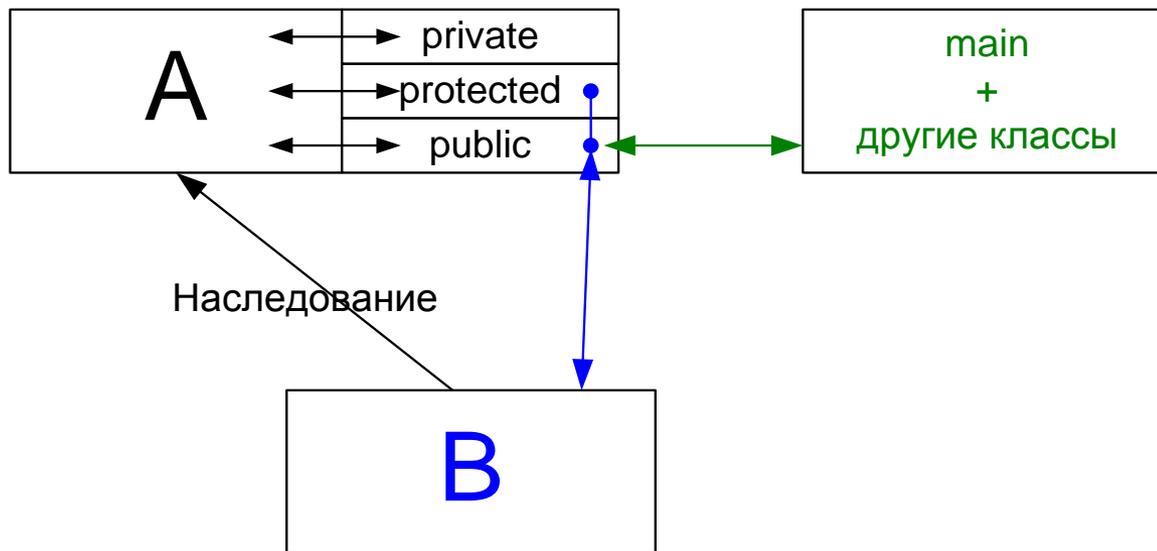
```
class event_t {
```

```
[...]
```

```
};
```

```
#endif
```

# Доступ к элементам класса



**public** (открытая) – видимы всем

**protected** (защищенная) – видимы самому классу, его подклассам и его друзьям

**private** (закрытая) – только самому классу и его друзьям

# Спецификаторы доступа в C++

```
class name_t {  
  
    public:  
        [...]  
  
    protected:  
        [...]  
  
    private:  
        [...]  
};  
  
class time_t {  
    public:  
  
        //! Устанавливает 00:00:00.  
        time_t();  
  
        //! Устанавливает время hour:00:00.  
        /*!  
        \param hour часы, 0..23.  
        */  
        time_t( int hour );  
  
        //! Устанавливает время hour:min:sec.  
        /*!  
        \param hour часы, 0..23.  
        \param min минуты, 0..59.  
        \param sec секунды, 0..59.  
  
        \note проверяет на корректность и  
        приводит к крайним значениям.  
        */  
        time_t( int hour, int min, int sec );  
  
        //! Печать состояния объекта.  
        void  
        display();  
  
    private:  
        int m_hour;  
        int m_min;  
        int m_sec;  
};
```

# Setter'ы и Getter'ы

```
class time_t {
public:

    // Setter hour.

    void
    set_hour( int hour ) {
        // Проверка на корректность.
        if ( ( hour > 0 ) && ( hour < 24 ) )
            m_hour = hour;
    }

    // Getter hour.

    int
    hour() {
        return m_hour;
    }

private:
    int m_hour;
    int m_min;
    int m_sec;
};
```

```
int main() {
    time_t time;

    time.set_hour(10);
    time.set_min(25);
    time.set_sec(40);

    std::cout <<
        time.hour() << ":" <<
        time.min() << ":" <<
        time.sec() <<
        std::endl;

};

10:25:40
```

# Изменение реализации

```
class time_t {
    public:

        void
        set_hour( int hour ) {
            // Проверка на корректность.
            if ( ( hour > 0 ) && ( hour < 24 ) )
                m_time = m_time % 3600 + m_hour * 3600;
        }

        int
        hour() {
            return (m_time / 60 / 60);
        }

        void
        display() {
            std::cout <<
                hour() << ":" <<
                min() << ":" <<
                sec() <<
                std::endl;
        }

    private:
        //! Время в секундах от начала суток.
        int m_time;
};
```