برنامه نویسی شیء گرا

Object Oriented Programming

برنامه نویسی شیء گرا

Object Oriented Programming

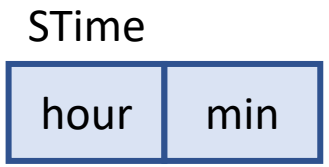
مهرین صارمی

پردیس فرزانگان

دانشگاه سمنان

struct

```
struct STime
{
    int hour;
    int min;
};
```

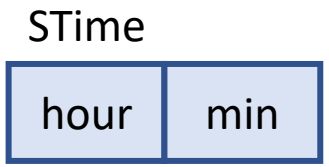


struct

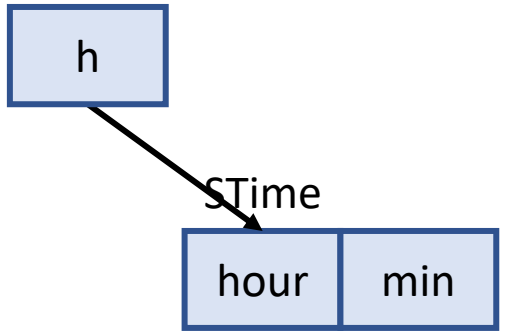
```
struct STime
{
    int hour;
    int min;
};


int main()
{
    STime t;
    t.hour = 15;
    t.min = 37;

    return 0;
}
```



struct




```
int main()
{
    STime t;
    int h;
    cin >> h;
    t.hour = h; // what if h=25 ?!

    return 0;
}
```

راه حل: دسترسی غیرمستقیم

```
void set_hour(STime& t, int h)
{
    if (h < 0 || h>23)
        h = 0;
    t.hour = h;
}

void set_min(STime& t, int m)
{
    if (m < 0 || m>59)
        m = 0;
    t.min = m;
}
```

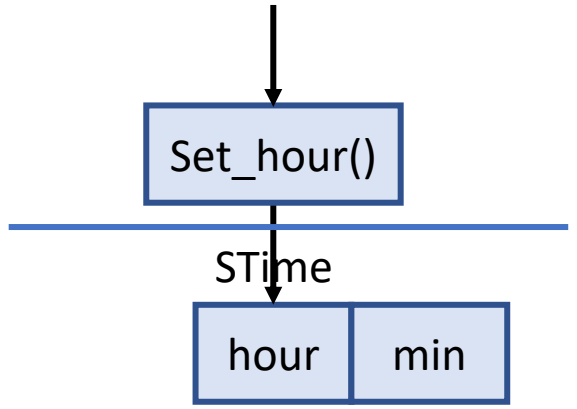
```
int main()
{
    STime t;
    int h;
    cin >> h;
    set_hour(t, h);

    return 0;
}
```

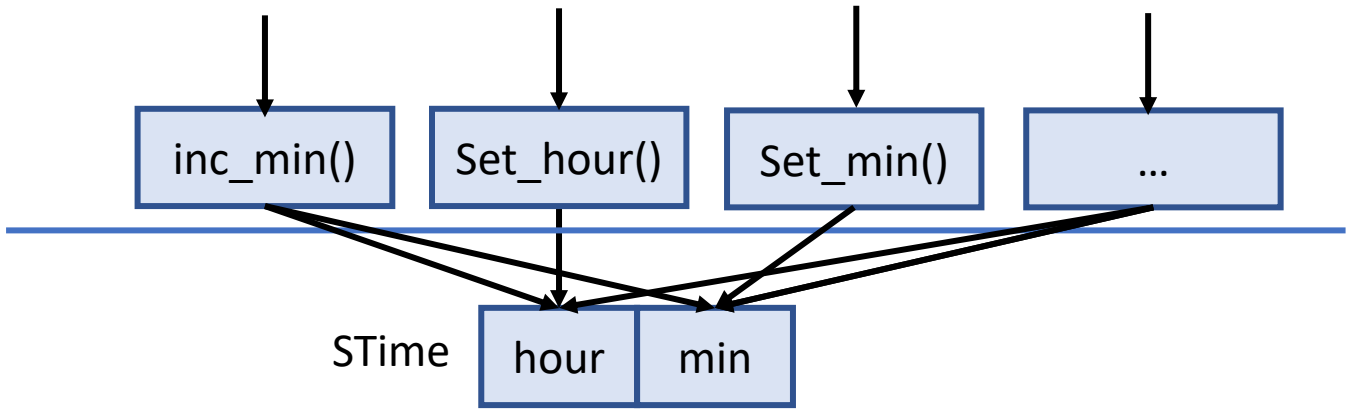
راه حل: دسترسی غیرمستقیم

```
int main()
{
    STime t;
    int h;
    cin >> h;
    set_hour(t, h);

    return 0;
}
```



راه حل: دسترسی غیرمستقیم

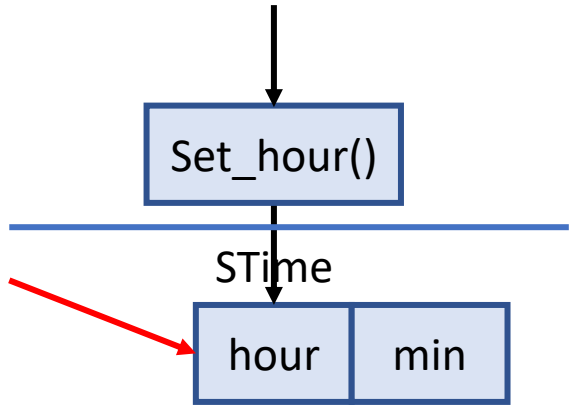


مشکل

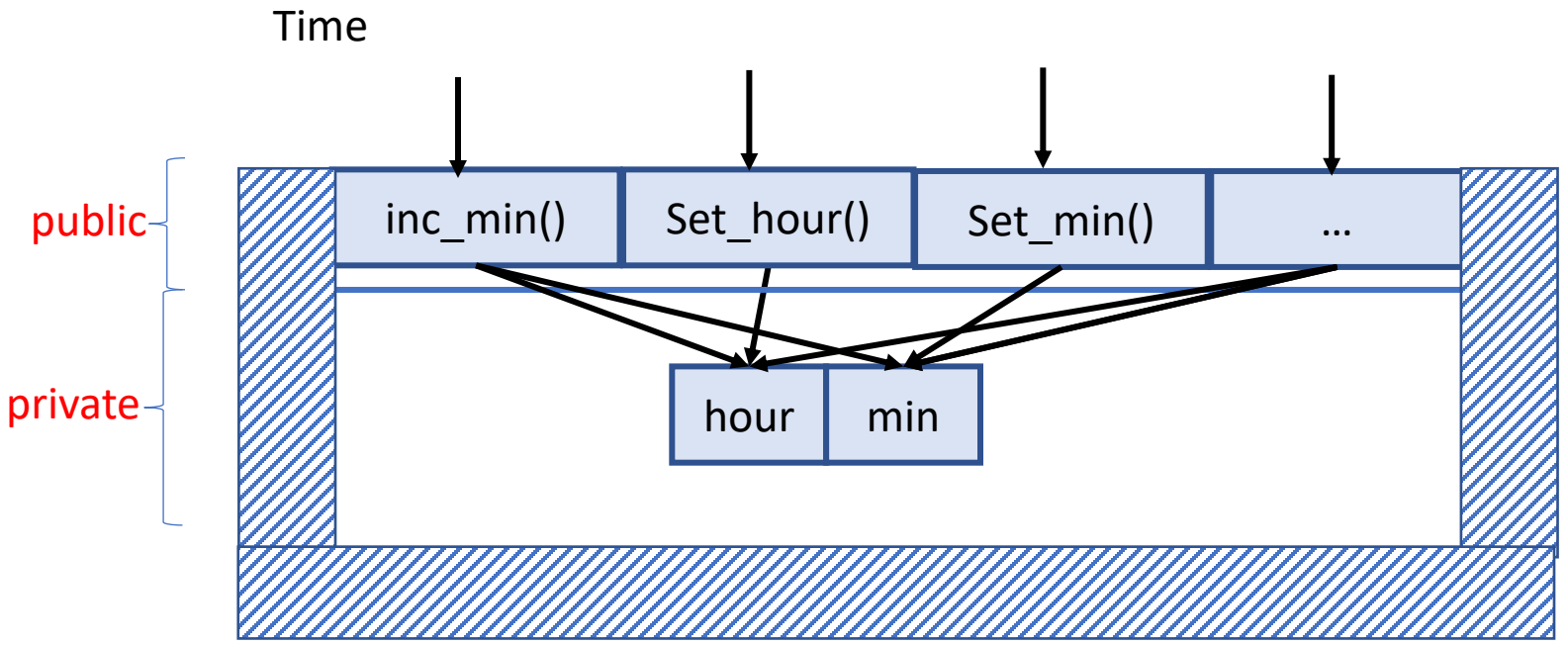
```
int main()
{
    STime t;
    int h;
    cin >> h;
    set_hour(t, h);
    t.hour = 25;

    return 0;
}
```

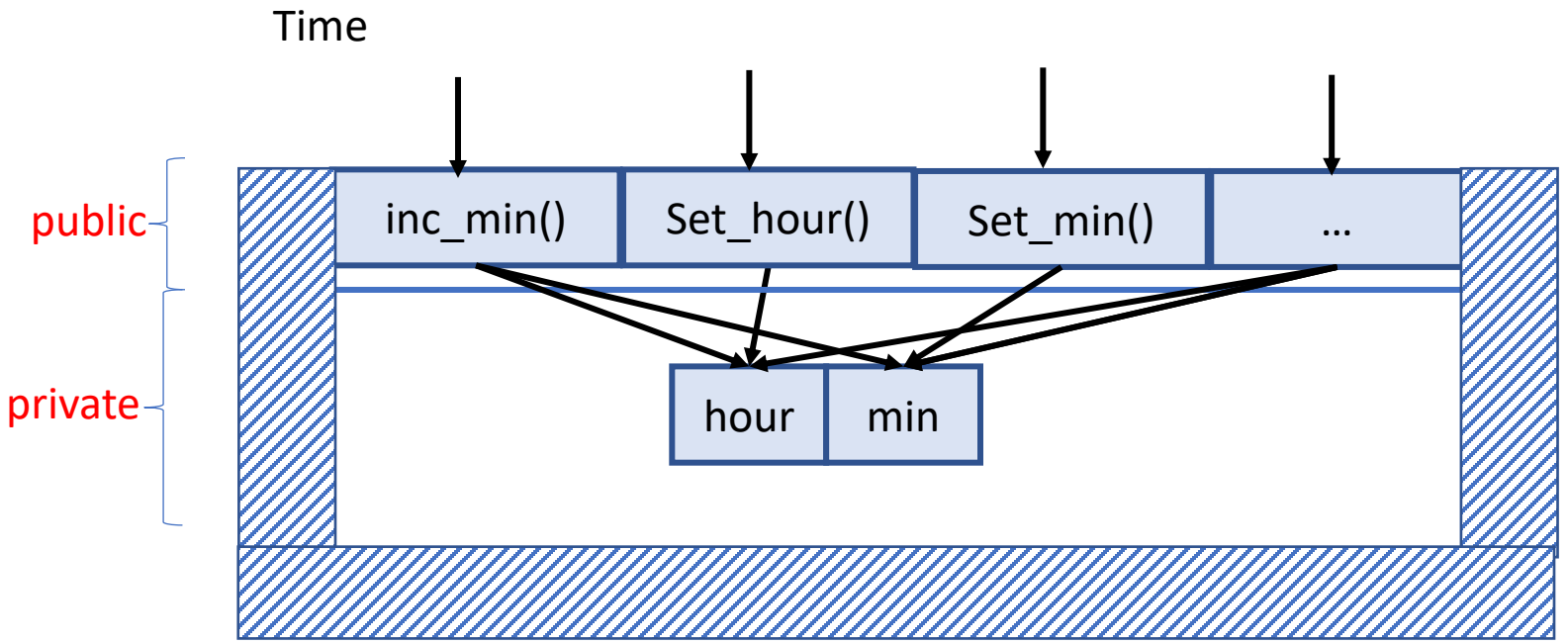
- اما همچنان امکان دسترسی مستقیم وجود دارد



راه حل: دسترسی عمومی و خصوصی



راه حل: دسترسی عمومی و خصوصی



- از دنیای خارج فقط به publicها دسترسی مستقیم داریم.
- اعضای public به اعضای private مستقیم دسترسی دارند

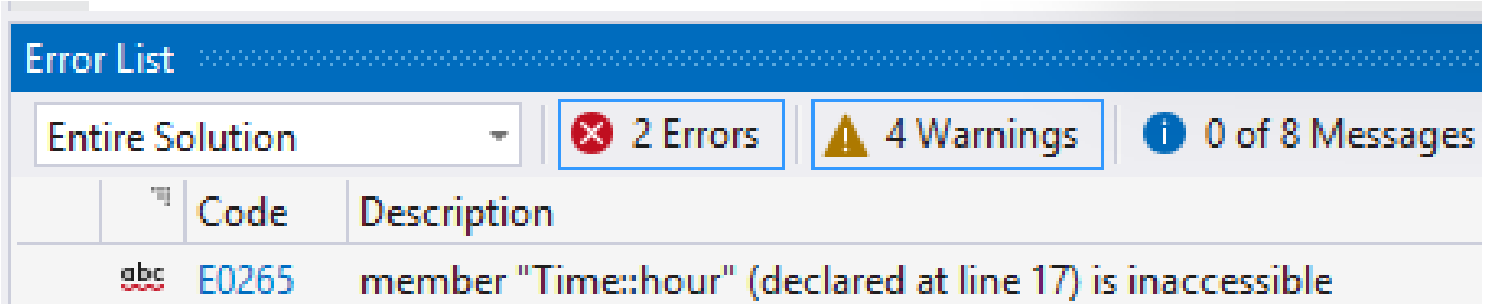
کلاس

```
class Time
{
private:
    int hour;
    int min;
public:
    void set_min(int m)
    {
        …
    }
    void set_hour(int h)
    {
        …
    }
};
```

خطای دسترسی

```
int main()
{
    Time t;
    int h;
    cin >> h;
    t.hour = h;

    return 0;
}
```



پیاده سازی توابع

```
class Time
{
private:
    int hour;
    int min;
public:
    void set_min(int m)
    {
        if (m < 0 || m > 59)
            m = 0;
        min = m;
    }
    void set_hour(int h)
    {
        if (h < 0 || h>23)
            h = 0;
        hour = h;
    }
};
```

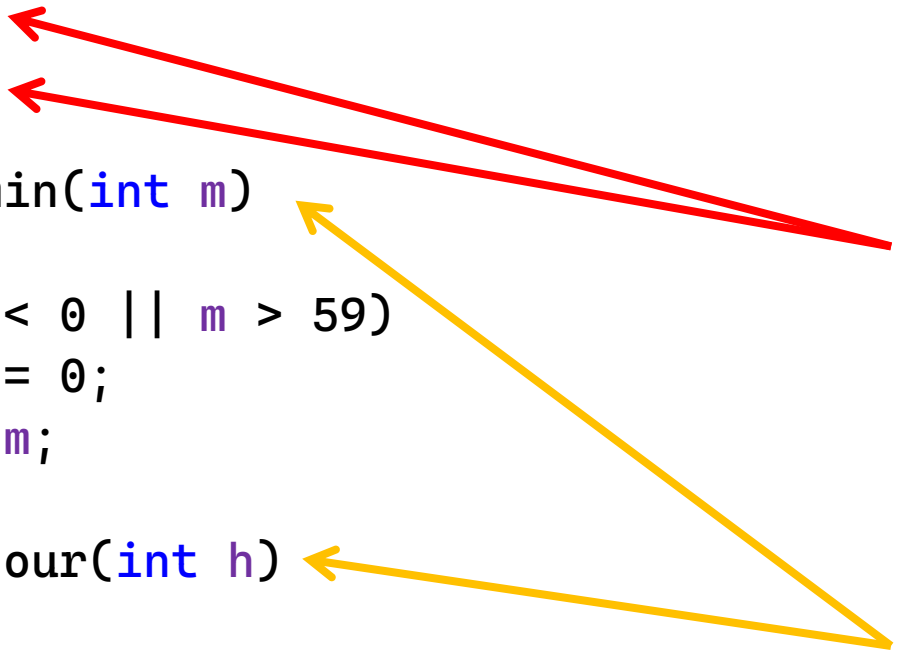
پیاده سازی توابع

- اعضای public به اعضای private مستقیم دسترسی دارند

```
class Time
{
private:
    int hour;
    int min;
public:
    void set_min(int m)
    {
        if (m < 0 || m > 59)
            m = 0;
        min = m;
    }
    void set_hour(int h)
    {
        if (h < 0 || h>23)
            h = 0;
        hour = h;
    }
};
```

اصطلاح

```
class Time
{
private:
    int hour;
    int min;
public:
    void set_min(int m)
    {
        if (m < 0 || m > 59)
            m = 0;
        min = m;
    }
    void set_hour(int h)
    {
        if (h < 0 || h>23)
            h = 0;
        hour = h;
    }
};
```



Data members

اعضای داده ای

Member functions (methods)

توابع عضو (متدها)

دسترسی عمومی و خصوصی

- نکته:
- اعضای داده ای لزوما private نیستند
- توابع عضو لزوما public نیستند

کپسوله سازی (encapsulation)

- بسته بندی داده و عملیات در کنار یکدیگر

- داده ها و عملیات (توابع) مربوط به آنها همه داخل کلاس قرار میگیرند.

پنهان سازی اطلاعات (information hiding)

- جزئیات پیاده سازی کلاس از دنیای خارج (مشتری) پنهان میشود
- کلاس تعدادی تابع را در اختیار جهان خارج میگذارد
- مشتری میتواند این توابع را فراخوانی کند بدون اینکه جزئیات پیاده سازی آنها را بداند
- این توابع رابط کلاس با مشتری هستند

اصطلاح: کلاس و شیء

```
class Time
{
…
};


int main()
{
    Time t;
    int h;
    cin >> h;
    t.hour = h;

    return 0;
}
```

class

Object
شیء

- به متغیری که از یک کلاس ساخته میشود شیء میگویند.
- کلاس مثل یک قالب یا الگو است که اشیاء از روی آن ساخته میشوند.

مقداردهی فیلدها (توابع Set)

```cpp
int main()
{
    Time t;
    int h;
    cin >> h;
    t.hour = h;

    return 0;
}
```

```cpp
int main()
{
    Time t;
    int h;
    cin >> h;
    t.set_hour(h);

    return 0;
}
```

خواندن فیلدها

```
int main()
{
    Time t;
    …
    cout << t.hour;

    return 0;
}
```

توابع get

```
class Time
{
private:
    int hour;
    int min;
public:
    int get_min()
    {
        return min;
    }
    int get_hour()
    {
        return hour;
    }
    …
};
```

توابع get

```
class Time
{
private:
    int hour;
    int min;
public:
    int get_min()
    {
        return min;
    }
    int get_hour()
    {
        return hour;
    }
    …
};
```

```
int main()
{
    Time t;
    …
    cout << t.hour;

    return 0;
}
```

```
int main()
{
    Time t;
    …
    cout << t.get_hour();

    return 0;
}
```

مقداردهی

- میخواهیم تابعی بنویسیم که همه فیلدها را مقداردهی کند.

```
class Time
{
private:
    int hour;
    int min;
public:
    void setup(int h, int m)
    {
        if (h < 0 || h>23)
            h = 0;
        hour = h;

        if (m < 0 || m > 59)
            m = 0;
        min = m;
    }
        …
};
```

استفاده مجدد

```
class Time
{
private:
    int hour;
    int min;
public:
    void setup(int h, int m)
    {
        set_hour(h);
        set_min(m);
    }
       …
};
```

- از توابعی که قبلا تعریف کرده بودیم استفاده کردیم.
- جلوگیری از تکرار

فراخوانی تابع

```
class Time
{
private:
    int hour;
    int min;
public:
    void setup(int h, int m)
    {
        set_hour(h);
        set_min(m);
    }
       …
};

int main()
{
    Time t;
    t.setup(5, 17);

    return 0;
}
```

تابع سازنده (constructor)

```
class Time
{
private:
    int hour;
    int min;
public:
    Time(int h, int m)
    {
        set_hour(h);
        set_min(m);
    }
…
}

int main()
{
    Time t(5, 17);

    return 0;
}
```

تابع سازنده (constructor)

```
class Time
{
private:
    int hour;
    int min;
public:
    Time(int h, int m)
    {
        set_hour(h);
        set_min(m);
    }
…
}

int main()
{
    Time t(5, 17);

    return 0;
}
```

- هر کلاس میتواند یک یا چند **تابع سازنده** داشته باشد
- تابع سازنده برای **مقداردهی اولیه** به کار میرود
- نام تابع سازنده با کلاس یکی است
- برای آن نوع خروجی تعریف نمیشود
- **موقع تعریف شیء، خود به خود** فراخوانی میشود.

- توجه: اینجا به جای تابع setup تابع سازنده را فراخواندیم.

فراخوانی تابع سازنده (constructor)

```
int main()
{
    Time t1(10, 20);
    Time t2{5, 17};
    return 0;
}
```

استفاده مجدد

```
class Time
{
private:
    int hour;
    int min;
public:
    Time(int h, int m)
    {
        setup(h, m);
    }
…
}

int main()
{
    Time t(5, 17);

    return 0;
}
```

توابع بیشتر (inc_min)

- یک دقیقه به time اضافه کن

```
class Time
{
…
public:
    void inc_min()
    {
        if (min == 59)
        {
            min = 0;
            if (hour == 23)
                hour = 0;
            else
                hour++;
        }
        else
        {
            min++;
        }
    }
…
};
```

توابع بیشتر (to_str)

- تبدیل به string

```
class Time
{
…
public:
string to_str()
    {
        string result = to_string(hour) + ":" + to_string(min);
        return result;
    }
…
};
```

آرگومان پیش فرض

- توابع عضو کلاس میتوانند مقدار پیش فرض داشته باشند.

- از جمله تابع سازنده

```
class Time
{
private:
    int hour;
    int min;
public:
    Time(int h=0, int m=0)
    {
        …
    }
…
}

int main()
{
    Time t1;          //default hour and min
    Time t2(5);       //default min
    Time t3(5, 15);
    return 0;
}
```

بیش از یک تابع سازنده

```
public:

    Time(int h=0, int m=0)
    {
        setup(h, m);
    }

    Time(string str)
    {
        if (str == "noon")
            setup(12, 0);
        else if (str == "midnight")
            setup(0, 0);
        else if (str == "night")
            setup(21, 0);
        else
            setup(0, 0);
    }
```

سازنده پیش فرض

- وقتی یک شیء بدون آرگومان ساخته می‌شود، باید یک تابع سازنده بدون آرگومان برای آن فراخوانی شود.
- یا باید یک سازنده بدون آرگومان وجود داشته باشد
- یا سازنده ای که همه آرگومانهایش مقدار پیش فرض داشته باشند

- به چنین سازنده هایی سازنده پیش فرض گفته میشود.

سازنده پیش فرض

- اگر برای کلاس **هیچ سازنده ای تعریف نشود** کامپایلر یک **سازنده پیش فرض** ایجاد میکند
- کار این سازنده چیست؟
  - مقداردهی به اعضای کلاس
  - به اعضای ساده مثل int, float, ... هیچ مقداری نمی‌دهد -> مقدار نامشخص خواهد بود
  - برای اعضایی که خود، از جنس یک کلاس هستند، سازنده پیش فرض آنها را فرامی‌خواند

Utility function

- تابعی که توسط توابع دیگر کلاس فراخوانی می‌شود
- برای استفاده توسط کلاینت کلاس نیست
- به طور خصوصی تعریف می‌شود

Utility function

- تابعی که توسط توابع دیگر کلاس فراخوانی می‌شود
- برای استفاده توسط کلاینت کلاس نیست
- به طور خصوصی تعریف می‌شود

```
class Date
{
private:
    int month;
    int day;
    int max_valid_day()
    {
        if (month <= 6)
            return 31;
        else
            return 30;
    }
public:
    ...
    void set_day(int day)
    {
        int max = max_valid_day();
        if (day > max)
            day = max;
        this->day = day;
    }
};
```

Initializer list

```
Time(int h=0, int m=0)
    :hour{h}, min{m}
    {
        if (h > 23 || h<0)
            hour = 0;
        ...
    }
```

The this pointer

```
Time(int h=0, int m=0)
    {
        hour = h;
        min = m;
    }
```

```
Time(int hour=0, int min=0)
  {
      hour = hour;
      min = min;
  }
```

The this pointer

```
Time(int h=0, int m=0)
    {
        hour = h;
        min = m;
    }
```

```
Time(int hour=0, int min=0)
  {
      hour = hour;
      min = min;
  }
```

```
Time(int hour=0, int min=0)
    {
        this->hour = hour;
        this->min = min;
    }
```

Friend function

```
class Time
{
    friend int test(Time);
private:
    …
public:

    …
};

int test(Time t)
{
    return t.hour;
}
```

Friend class

```
class Time
{
    friend class X;
private:
    …
public:

    …
};
```

- از داخل کلاس X میتوان به اعضای خصوصی کلاس Time دسترسی داشت
- دوستی رابطه یک طرفه است!

Friend class

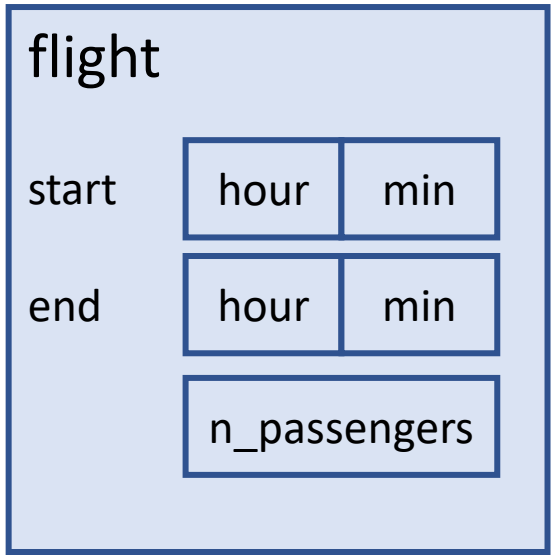
```
class Time
{
    friend class X;
private:
    …
public:

    …
};
```

- از داخل کلاس X میتوان به اعضای خصوصی کلاس Time دسترسی داشت
- دوستی رابطه یک طرفه است!
- Time به اعضای خصوصی X دسترسی ندارد.
  مگر آنکه X هم Time را دوست خود اعلام کند.

Composition

```
class flight
{
private:
    Time start;
    Time end;
    int n_passengers;
 };
```



Composition

```
class flight
{
private:
    Time start;
    Time end;
    int n_passengers;

public:
    flight(Time s, Time e, int n)
        :start{s}, end{e}, n_passengers{n}
    {

    }
};
```

```
class Time
{
    …
    int get_hour()
    {
        return this->hour;
    }
};
```

جداسازی اعلان از پیاده سازی

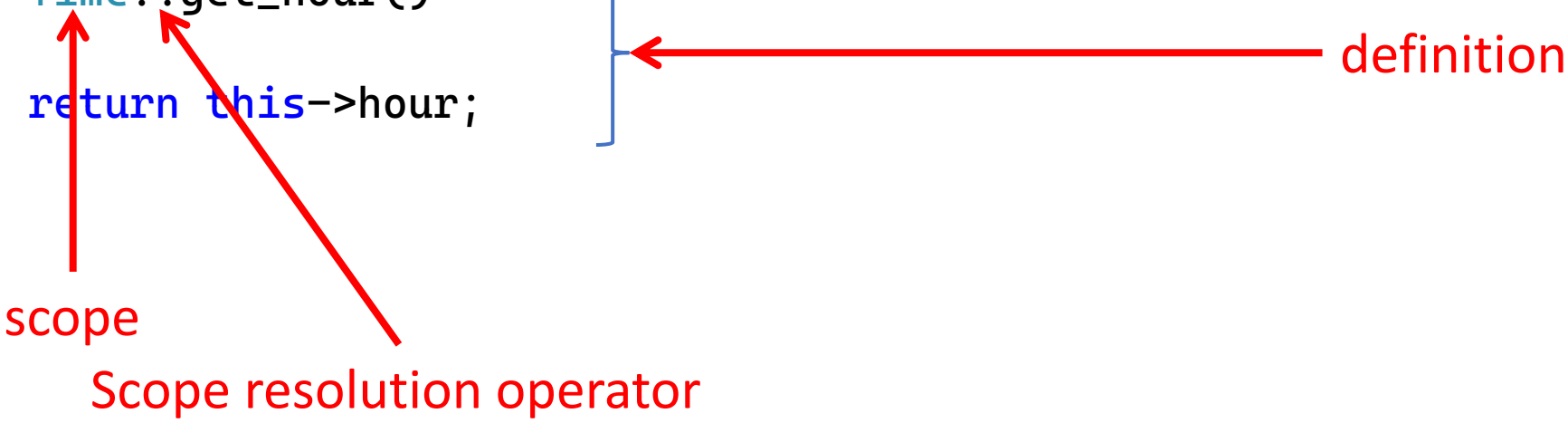
```
class Time
{
    …
    int get_hour();
};

int Time::get_hour()
{
    return this->hour;
}
```

جداسازی اعلان از پیاده سازی

```
class Time
{
    …
    int get_hour();
};

int Time::get_hour()
{
    return this->hour;
}
```



Header files

time.h

```
class Time
{
private:
    int hour;
    int min;
public:

    Time(int h = 0, int m = 0)
    {
        setup(h, m);
    }


    …
};
```

main.cpp

```
#include "time.h"

int main()
{
    Time t1(12, 35);

    return 0;
}
```

Separate header & source

time.h

```
class Time
{
private:
    int hour;
    int min;
public:

    Time(int h = 0, int m = 0);
    …
};
```

time.cpp

```
#include "time.h"

Time::Time(int h, int m)
{
    setup(h, m);
}

void Time::setup(int h, int m)
{
    set_hour(h);
    set_min(m);
}
…
```

Separate header & source

time.h

```
class Time
{
…
};
```

time.cpp

```
#include "time.h"

Time::Time(int h, int m)
{
    setup(h, m);
}
…
```

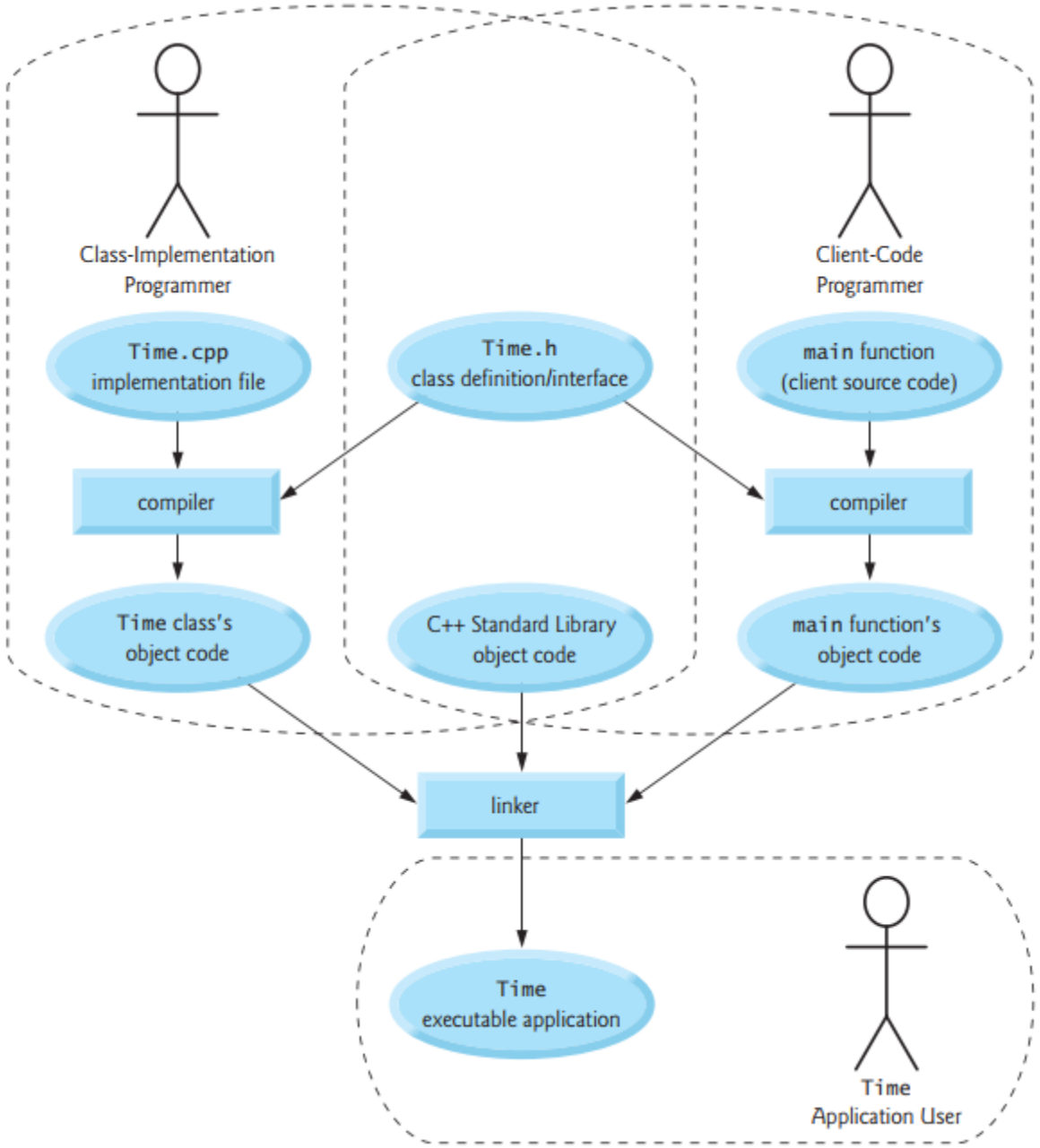
main.cpp

```
#include "time.h"

int main()
{
    Time t1(12, 35);

    return 0;
}
```

Compile & link



Include guard

```
#ifndef TIME_H
#define TIME_H

class Time
{
…

};

#endif
```

Include guard

- روش دوم

```
#pragma once


class Time
{
…

};
```