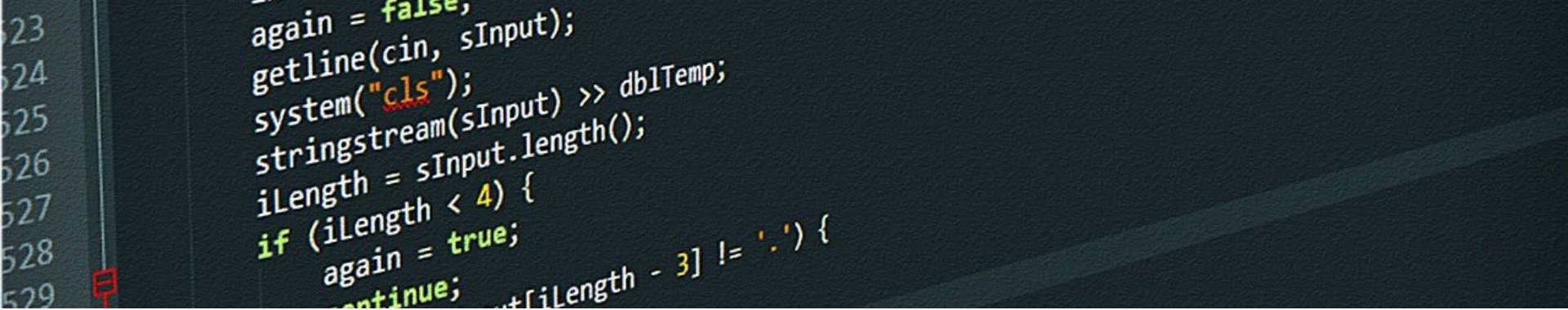

Thomas

C23-06.1 Static members

Advanced algorithms and programming

v5

Hochschule für Technik und Wirtschaft Berlin
University of Applied Sciences

Static class elements

Member variables

- Class member variables declared with the keyword `static` are called *class variables* or *static member variables*
- All objects (instances) of the class share the same variable
- Memory for the variable is allocated only once in the data segment
  For this purpose, the variable must be defined, e.g.:

```
Type Class_name::Class_variable;
```

- Static members can also be used independently of an object, if declared `public`
  (access via Scope operator ::) ← **Good style**
- Static members can be used as replacement for 'C'-style global variables

- Initialization must be done outside the class using the scope operator, e.g. in combination with the variable definition

```
Type Class_name::Class_variable {Init_value};
```

Static class elements

Member variables – example

Point.h

```
#pragma once
#include <string>

class Point
{
public:
    Point(int x = 0, int y = 0);

    std::string toString() const;

    int getNumPoints() const;

    static int s_color; // Negative example for effect of 'static

private:
    int m_x, m_y;
    static int s_numPoints; // useful application
};
```

Point.cpp

```
#include <iostream>
#include <sstream>
#include <iomanip>
#include "Point.h"

// Initialization of static variables
int Point::s_numPoints = 0;
int Point::s_color = 0xFFFFFF;

Point::Point(int x, int y) : m_x(x), m_y(y)
{
    s_numPoints += 1;
}

std::string Point::toString() const
{
    std::stringstream stream;
    stream << "(" << m_x << "," << m_y << ")" << " : #"
        << std::setw(6) << std::setfill('0')
        << std::uppercase << std::hex << s_color;
    return stream.str();
}

int Point::getNumPoints() const { return s_numPoints; }
```

Static class elements

Member variables – example

```
Number of points: 1
Number of points: 3
P1(10,25) : #FFFFFF
P2(30,60) : #FFFFFF
P3(20,10) : #FFFFFF
P1(10,25) : #FF0000
P2(30,60) : #FF0000
P3(20,10) : #FF0000
P1(10,25) : #00FF00
P2(30,60) : #00FF00
P3(20,10) : #00FF00
```

Main.cpp

```cpp
#include <iostream>
#include "Point.h"

int main()
{
    Point p1(10, 25);
    std::cout << "Number of points: " << p1.getNumPoints() << std::endl;
    Point p2(30, 60);
    Point p3(20, 10);
    std::cout << "Number of points: " << p1.getNumPoints() << std::endl;

    std::cout << "P1" << p1.toString() << "\nP2" << p2.toString() << "\nP3" << p3.toString() << std::endl;
    p1.s_color = 0xFF0000;
    std::cout << "P1" << p1.toString() << "\nP2" << p2.toString() << "\nP3" << p3.toString() << std::endl;
    Point::s_color = 0x00FF00; // Access via scope operator
    std::cout << "P1" << p1.toString() << "\nP2" << p2.toString() << "\nP3" << p3.toString() << std::endl;
    return 0;
}
```

Static class elements

Member functions

- Member functions declared in a class with the keyword `static` can be called independently of an object (access via the scope operator `::`) ← **Good style**
- Static member functions can be called without an object of the class already existing (for example to implement the "factory" pattern)

- Attention:
  - The **`this`** pointer is not available in static member functions
  - Static member functions cannot use (normal) member variables or member functions of the class (exception: static variables and functions)

Static class elements

Member functions – example

Point.h

```
#pragma once
#include <string>

class Point
{
public:

    Point(int x = 0, int y = 0);

    std::string toString() const;

    // 'static' function cannot be 'const'!
    static int getNumPoints();

private:
    int m_x, m_y;
    static int s_numPoints;
};
```

Point.cpp

```
#include <iostream>
#include <sstream>
#include "Point.h"

// Initialization of static variables
int Point::s_numPoints = 0;

Point::Point(int x, int y) : m_x(x), m_y(y)
{
    s_numPoints += 1;
}

std::string Point::toString() const
{
    std::stringstream stream;
    stream << "(" << m_x << "," << m_y << ")";
    return stream.str();
}

// Do not use 'static' in the definiton!
int Point::getNumPoints()
{
    return s_numPoints;
}
```

Static class elements

Member functions – example

Main.cpp

```cpp
#include <iostream>
#include "Point.h"

int main()
{
    Point p1(10, 25);

    // Access to static function via Scope operator:
    std::cout << "Number of points: " << Point::getNumPoints() << std::endl;
    Point p2(30, 60);
    Point p3(20, 10);

    // Also uses the point operator:
    std::cout << "Number of points: " << p3.getNumPoints() << std::endl;

    std::cout << "P1" << p1.toString() << "\nP2" << p2.toString()
        << "\nP3" << p3.toString() << std::endl;

}
```

```
Number of points: 1
Number of points: 3
P1(10,0)
P2(30,60)
P3(20,10)
```

Static class elements

"Best Practice" for static class elements

- Always access via class name and scope operator (without object) for access
  This immediately shows that the element is static!

- Static member variables are, e.g., useful for:
  - Settings that should apply to all objects of a class
  - Reference Counter

- Static member functions useful for:
  - To create instances of a class (Factory pattern)
  - For all functions that do not use non-static member variables

Design patterns
Factory method pattern

- Intent
  - Define an interface for creating an object, but let subclasses decide which class to instantiate

- Approach
  - A superclass specifies all standard and generic behavior (using pure virtual "placeholders" for creation steps)
  - Creation details are left to subclasses (that may be supplied later in the design process)

- Advantage
  - Factory Method pattern makes a design more customizable and only a little more complicated

Source: https://sourcemaking.com/designpatterns/factory_method/cpp/1

Design patterns

Factory method pattern – example

Stooge.h

```
#pragma once
#include <iostream>

class Stooge
{
public:
    // Factory Method
    static Stooge* make_stooge(int choice);
    virtual void slap_stick() = 0;
};
```

```
class Larry : public Stooge
{
public:
    void slap_stick()
    {    std::cout << "Larry: jump\n"; }
};

class Moe : public Stooge
{
public:
    void slap_stick()
    {    std::cout << "Moe: stumble\n"; }
};

class Curly : public Stooge
{
public:
    void slap_stick()
    {    std::cout << "Curly: fall over\n"; }
};
```

Source: https://sourcemaking.com/designpatterns/factory_method/cpp/1

Design patterns

Factory method pattern – example

Stooge.cpp

```cpp
#include "Stooge.h"

Stooge* Stooge::make_stooge(int choice)
{
    switch (choice)
    {
    case 1: return new Larry; break;
    case 2: return new Moe; break;
    default: return new Curly; break;
    }
}
```

Main.cpp

```cpp
#include <vector>
#include "Stooge.h"

int main()
{
    std::vector<Stooge*> roles;
    int choice;
    while (true)
    {
        std::cout << "Choose from Larry(1) Moe(2) Curly(3),
press (0) to go: ";
        std::cin >> choice;
        if (choice == 0)
            break;
        roles.push_back(Stooge::make_stooge(choice));
    }
    for (int i = 0; i < roles.size(); i++)
        roles[i]->slap_stick();
    for (int i = 0; i < roles.size(); i++)
        delete roles[i];
}
```

```
Choose from Larry(1) Moe(2) Curly(3), press (0) to go: 2
Choose from Larry(1) Moe(2) Curly(3), press (0) to go: 3
Choose from Larry(1) Moe(2) Curly(3), press (0) to go: 1
Choose from Larry(1) Moe(2) Curly(3), press (0) to go: 0
Moe: stumble
Curly: fall over
Larry: jump
```

Source: https://sourcemaking.com/designpatterns/factory_method/cpp/1

Hochschule für Technik
und Wirtschaft Berlin

University of Applied Sciences

www.htw-berlin.de