STATIC MEMBERS AND FRIENDS

Lecturer：陈笑沙

TABLE OF CONTENTS

7.1 WHY DO WE NEED STATIC MEMBERS

WHAT ARE STATIC MEMBERS?

7.1 WHY DO WE NEED STATIC MEMBERS

WHAT ARE STATIC MEMBERS?

- When a class member is declared with the keyword `static`, it becomes a static member.

7.1 WHY DO WE NEED STATIC MEMBERS

WHAT ARE STATIC MEMBERS?

- When a class member is declared with the keyword `static`, it becomes a static member.
- Static members are divided into:

7.1 WHY DO WE NEED STATIC MEMBERS

WHAT ARE STATIC MEMBERS?

- When a class member is declared with the keyword `static`, it becomes a static member.
- Static members are divided into:
  - Static member variables

7.1 WHY DO WE NEED STATIC MEMBERS

WHAT ARE STATIC MEMBERS?

- When a class member is declared with the keyword `static`, it becomes a static member.
- Static members are divided into:
  - Static member variables
  - Static member functions

7.1 WHY DO WE NEED STATIC MEMBERS

Static member variable definition and initialization

```
1  class A {
2  public:
3      static int property;
4  };
5
6  int A::property = 1;
```

7.1 WHY DO WE NEED STATIC MEMBERS

Static member variable definition and initialization

```
1  class A {
2  public:
3      static int property;
4  };
5
6  int A::property = 1;
```

7.1 WHY DO WE NEED STATIC MEMBERS

Static member variable definition and initialization

```
1  class A {
2  public:
3      static int property;
4  };
5
6  int A::property = 1;
```

7.1 WHY DO WE NEED STATIC MEMBERS

Static member variable definition and initialization

```
1  class A {
2  public:
3      static int property;
4  };
5
6  int A::property = 1;
```

7.1 WHY DO WE NEED STATIC MEMBERS

Static member variable characteristics

7.1 WHY DO WE NEED STATIC MEMBERS

Static member variable characteristics

- All objects of the class share the static member variable, so regardless of how many objects of the class are created, there is only one copy of the static member variable.

7.1 WHY DO WE NEED STATIC MEMBERS

Static member variable characteristics

- All objects of the class share the static member variable, so regardless of how many objects of the class are created, there is only one copy of the static member variable.
- The static member variable belongs to the class, not to a specific object.

7.1 WHY DO WE NEED STATIC MEMBERS

Static member variable characteristics

- All objects of the class share the static member variable, so regardless of how many objects of the class are created, there is only one copy of the static member variable.
- The static member variable belongs to the class, not to a specific object.
- The lifecycle of the static member variable is the same as that of a global variable.

7.1 WHY DO WE NEED STATIC MEMBERS

Some attributes describe the commonality of all objects

```
1  class EnemySlime {
2      int maxHealth;
3      float maxSpeed;
4  };
```

7.1 WHY DO WE NEED STATIC MEMBERS

Some attributes describe the commonality of all objects

```
1  class EnemySlime {
2      int maxHealth;
3      float maxSpeed;
4  };
```

But so far, the syntax only allows classes to describe object properties, not shared properties.

7.1 WHY DO WE NEED STATIC MEMBERS

7.1 WHY DO WE NEED STATIC MEMBERS

Can we use global variables?

7.1 WHY DO WE NEED STATIC MEMBERS

Can we use global variables?

```
1  int slimeMaxHealth;
2  float slimeMaxSpeed;
3  class Slime {...};
```

7.1 WHY DO WE NEED STATIC MEMBERS

Can we use global variables?

```
1  int slimeMaxHealth;
2  float slimeMaxSpeed;
3  class Slime {...};
```

- Easy to cause naming conflicts

7.1 WHY DO WE NEED STATIC MEMBERS

Can we use global variables?

```
1  int slimeMaxHealth;
2  float slimeMaxSpeed;
3  class Slime {...};
```

- Easy to cause naming conflicts
- Unable to control accessibility (code is visible anywhere)

7.1 WHY DO WE NEED STATIC MEMBERS

Can we use global variables?

```
1  int slimeMaxHealth;
2  float slimeMaxSpeed;
3  class Slime {...};
```

- Easy to cause naming conflicts
- Unable to control accessibility (code is visible anywhere)
- Avoid using global variables (not a good coding style)

7.2 HOW TO USE STATIC MEMBERS

7.2 HOW TO USE STATIC MEMBERS

- Static data members also have public and private attributes

7.2 HOW TO USE STATIC MEMBERS

- Static data members also have public and private attributes
  - Only public attribute static data members can be accessed outside the class

7.2 HOW TO USE STATIC MEMBERS

- Static data members also have public and private attributes
  - Only public attribute static data members can be accessed outside the class
  - All attribute static data members can be accessed inside the class

7.2 HOW TO USE STATIC MEMBERS

- Static data members also have public and private attributes
  - Only public attribute static data members can be accessed outside the class
  - All attribute static data members can be accessed inside the class
- Static data members belong to the class

7.2 HOW TO USE STATIC MEMBERS

- Static data members also have public and private attributes
  - Only public attribute static data members can be accessed outside the class
  - All attribute static data members can be accessed inside the class
- Static data members belong to the class
  - Access method: ClassName::StaticMember

7.2 HOW TO USE STATIC MEMBERS

- Static data members also have public and private attributes
  - Only public attribute static data members can be accessed outside the class
  - All attribute static data members can be accessed inside the class
- Static data members belong to the class
  - Access method: ClassName::StaticMember
- Class static data members can also be accessed when no objects exist

7.3 STATIC MEMBER VARIABLES

```
1  class Slime {
2      static int maxHealth;
3      static int maxSpeed;
4      int health;
5      int speed;
6  };
```

7.3 STATIC MEMBER VARIABLES

```
1  class Slime {
2      static int maxHealth;
3      static int maxSpeed;
4      int health;
5      int speed;
6  };
```

7.3 STATIC MEMBER VARIABLES

```
1  class Slime {
2      static int maxHealth;
3      static int maxSpeed;
4      int health;
5      int speed;
6  };
```

7.3 STATIC MEMBER VARIABLES

```
1  class Slime {
2      static int maxHealth;
3      static int maxSpeed;
4      int health;
5      int speed;
6  };
```

7.3 STATIC MEMBER VARIABLES

```
 1  class Slime {
 2  public:
 3      void heal(int amount) {
 4          health = std::min(health + amount, maxHealth);
 5      }
 6
 7      void acc(int amount) {
 8          speed += amount;
 9          if (speed > maxSpeed)
10              speed = maxSpeed;
11          else if (speed < -maxSpeed)
12              speed = -maxSpeed;
13      }
14  };
```

7.3 STATIC MEMBER VARIABLES

```
1  class Slime {
2  public:
3      void heal(int amount) {
4          health = std::min(health + amount, maxHealth);
5      }
6
7      void acc(int amount) {
8          speed += amount;
9          if (speed > maxSpeed)
10             speed = maxSpeed;
11         else if (speed < -maxSpeed)
12             speed = -maxSpeed;
13     }
14 };
```

7.3 STATIC MEMBER VARIABLES

```
1  class Slime {
2  public:
3      void heal(int amount) {
4          health = std::min(health + amount, maxHealth);
5      }
6
7      void acc(int amount) {
8          speed += amount;
9          if (speed > maxSpeed)
10             speed = maxSpeed;
11         else if (speed < -maxSpeed)
12             speed = -maxSpeed;
13     }
14 };
```

7.3 STATIC MEMBER VARIABLES

```
1  class Slime {
2  public:
3      void heal(int amount) {
4          health = std::min(health + amount, maxHealth);
5      }
6
7      void acc(int amount) {
8          speed += amount;
9          if (speed > maxSpeed)
10             speed = maxSpeed;
11         else if (speed < -maxSpeed)
12             speed = -maxSpeed;
13     }
14 };
```

7.3 STATIC MEMBER VARIABLES

```
 1  class Slime {
 2  public:
 3      void heal(int amount) {
 4          health = std::min(health + amount, maxHealth);
 5      }
 6
 7      void acc(int amount) {
 8          speed += amount;
 9          if (speed > maxSpeed)
10              speed = maxSpeed;
11          else if (speed < -maxSpeed)
12              speed = -maxSpeed;
13      }
14  };
```

7.4 STATIC MEMBER FUNCTIONS

Definition of static member functions:

```
1  class A {
2    static void func() { ...}
3  };
```

7.4 STATIC MEMBER FUNCTIONS

Look at the following example:

```
1  class Vector2D {
2  private:
3      double x, y;
4  public:
5      Vector2D(x = 0.0, y = 0.0) : x(x), y(y) {}
6      Vector2D& add(const Vector2D& other) {
7          x += other.x;
8          y += other.y;
9          return *this; // why this?
10     }
11
12     std::string to_string() const {
13         std::ostringstream oss;
14         oss << "{" << x << ", " << y << "}";
15         return oss.str();
```

7.4 STATIC MEMBER FUNCTIONS

Look at the following example:

```
1  class Vector2D {
2  private:
3      double x, y;
4  public:
5      Vector2D(x = 0.0, y = 0.0) : x(x), y(y) {}
6      Vector2D& add(const Vector2D& other) {
7          x += other.x;
8          y += other.y;
9          return *this; // why this?
10     }
11
12     std::string to_string() const {
13         std::ostringstream oss;
14         oss << "{" << x << ", " << y << "}";
15         return oss.str();
```

7.4 STATIC MEMBER FUNCTIONS

Look at the following example:

```
1  class Vector2D {
2  private:
3      double x, y;
4  public:
5      Vector2D(x = 0.0, y = 0.0) : x(x), y(y) {}
6      Vector2D& add(const Vector2D& other) {
7          x += other.x;
8          y += other.y;
9          return *this; // why this?
10     }
11
12     std::string to_string() const {
13         std::ostringstream oss;
14         oss << "{" << x << ", " << y << "}";
15         return oss.str();
```

7.4 STATIC MEMBER FUNCTIONS

Look at the following example:

```
1  class Vector2D {
2  private:
3      double x, y;
4  public:
5      Vector2D(x = 0.0, y = 0.0) : x(x), y(y) {}
6      Vector2D& add(const Vector2D& other) {
7          x += other.x;
8          y += other.y;
9          return *this; // why this?
10     }
11
12     std::string to_string() const {
13         std::ostringstream oss;
14         oss << "{" << x << ", " << y << "}";
15         return oss.str();
```

7.4 STATIC MEMBER FUNCTIONS

Look at the following example:

```
2  private:
3      double x, y;
4  public:
5      Vector2D(x = 0.0, y = 0.0) : x(x), y(y) {}
6      Vector2D& add(const Vector2D& other) {
7          x += other.x;
8          y += other.y;
9          return *this; // why this?
10     }
11
12     std::string to_string() const {
13         std::ostringstream oss;
14         oss << "{" << x << ", " << y << "}";
15         return oss.str();
16     }
```

7.4 STATIC MEMBER FUNCTIONS

Look at the following example:

```
    double x, y;
public:
    Vector2D(x = 0.0, y = 0.0) : x(x), y(y) {}
    Vector2D& add(const Vector2D& other) {
        x += other.x;
        y += other.y;
        return *this; // why this?
    }

    std::string to_string() const {
        std::ostringstream oss;
        oss << "{" << x << ", " << y << "}";
        return oss.str();
    }
};
        return oss.str();
```

7.4 STATIC MEMBER FUNCTIONS

Returning itself can achieve chain calls:

```
1  #include "vector2d.hpp"
2  #include <iostream>
3
4  using std::cout;
5  using std::endl;
6
7  int main(int argc, char **argv) {
8    Vector2D v;
9    v.add({1, 2}).add({3, 4});
10   cout << v.to_string() << endl;
11   return 0;
12 }
```

7.4 STATIC MEMBER FUNCTIONS

Returning itself can achieve chain calls:

```
1  #include "vector2d.hpp"
2  #include <iostream>
3
4  using std::cout;
5  using std::endl;
6
7  int main(int argc, char **argv) {
8    Vector2D v;
9    v.add({1, 2}).add({3, 4});
10   cout << v.to_string() << endl;
11   return 0;
12 }
```

7.4 STATIC MEMBER FUNCTIONS

Returning itself can achieve chain calls:

```
1  #include "vector2d.hpp"
2  #include <iostream>
3
4  using std::cout;
5  using std::endl;
6
7  int main(int argc, char **argv) {
8    Vector2D v;
9    v.add({1, 2}).add({3, 4});
10   cout << v.to_string() << endl;
11   return 0;
12 }
```

7.4 STATIC MEMBER FUNCTIONS

Returning itself can achieve chain calls:

```
1  #include "vector2d.hpp"
2  #include <iostream>
3
4  using std::cout;
5  using std::endl;
6
7  int main(int argc, char **argv) {
8    Vector2D v;
9    v.add({1, 2}).add({3, 4});
10   cout << v.to_string() << endl;
11   return 0;
12 }
```

7.4 STATIC MEMBER FUNCTIONS

Returning itself can achieve chain calls:

```
1  #include "vector2d.hpp"
2  #include <iostream>
3
4  using std::cout;
5  using std::endl;
6
7  int main(int argc, char **argv) {
8    Vector2D v;
9    v.add({1, 2}).add({3, 4});
10   cout << v.to_string() << endl;
11   return 0;
12 }
```

7.4 STATIC MEMBER FUNCTIONS

Returning itself can achieve chain calls:

```
 1  #include "vector2d.hpp"
 2  #include <iostream>
 3
 4  using std::cout;
 5  using std::endl;
 6
 7  int main(int argc, char **argv) {
 8    Vector2D v;
 9    v.add({1, 2}).add({3, 4});
10    cout << v.to_string() << endl;
11    return 0;
12  }
```

7.4 STATIC MEMBER FUNCTIONS

Returning itself can achieve chain calls:

```
1  #include "vector2d.hpp"
2  #include <iostream>
3
4  using std::cout;
5  using std::endl;
6
7  int main(int argc, char **argv) {
8    Vector2D v;
9    v.add({1, 2}).add({3, 4});
10   cout << v.to_string() << endl;
11   return 0;
12 }
```

7.4 STATIC MEMBER FUNCTIONS

But does addition have to be performed on one vector to another?

```
1  class Vector2D {
2    static Vector2D add(const Vector2D& v1, const Vector2D& v2) {
3      Vector2D v(v1.x + v2.x, v1.y + v2.y);
4      return v;
5    }
6  };
```

7.4 STATIC MEMBER FUNCTIONS

But does addition have to be performed on one vector to another?

```
1  class Vector2D {
2    static Vector2D add(const Vector2D& v1, const Vector2D& v2) {
3      Vector2D v(v1.x + v2.x, v1.y + v2.y);
4      return v;
5    }
6  };
```

7.4 STATIC MEMBER FUNCTIONS

But does addition have to be performed on one vector to another?

```
1  class Vector2D {
2    static Vector2D add(const Vector2D& v1, const Vector2D& v2) {
3      Vector2D v(v1.x + v2.x, v1.y + v2.y);
4      return v;
5    }
6  };
```

7.4 STATIC MEMBER FUNCTIONS

But does addition have to be performed on one vector to another?

```
1  #include "vector2d.hpp"
2  #include <iostream>
3
4  using std::cout;
5  using std::endl;
6
7  int main(int argc, char **argv) {
8    Vector2D v;
9    v.add({1, 2}).add({3, 4});
10   cout << v.to_string() << endl;
11
12   Vector2D v2{100, 200};
13   Vector2D v3 = Vector2D::add(v, v2);
14   cout << v3.to_string() << endl;
15   return 0;
```

7.4 STATIC MEMBER FUNCTIONS

But does addition have to be performed on one vector to another?

```
#include <iostream>

using std::cout;
using std::endl;

int main(int argc, char **argv) {
  Vector2D v;
  v.add({1, 2}).add({3, 4});
  cout << v.to_string() << endl;

  Vector2D v2{100, 200};
  Vector2D v3 = Vector2D::add(v, v2);
  cout << v3.to_string() << endl;
  return 0;
}
```

7.4 STATIC MEMBER FUNCTIONS

Another example: /example/lec07/itemManage

7.5 WHY DO WE NEED FRIENDS

Think about a question:

If there is a List class and a Node class, how to implement list.append(Node&)

7.5 WHY DO WE NEED FRIENDS

7.5 WHY DO WE NEED FRIENDS

- Node class has next as a private member.

7.5 WHY DO WE NEED FRIENDS

- Node class has next as a private member.
- List class has head as a private member.

7.5 WHY DO WE NEED FRIENDS

- Node class has next as a private member.
- List class has head as a private member.
- However, we need to modify next of node in list.

7.5 WHY DO WE NEED FRIENDS

7.5 WHY DO WE NEED FRIENDS

- The purpose of encapsulation is to achieve information hiding.

7.5 WHY DO WE NEED FRIENDS

- The purpose of encapsulation is to achieve information hiding.
- Private members of an object can only be accessed by its own members. When external objects or functions need to access the private members of a class, they can only do so indirectly through the public members provided by the class.

7.5 WHY DO WE NEED FRIENDS

- The purpose of encapsulation is to achieve information hiding.
- Private members of an object can only be accessed by its own members. When external objects or functions need to access the private members of a class, they can only do so indirectly through the public members provided by the class.
- C++ provides a friend mechanism to break through the boundaries of privatization, meaning that a class's friend can access its private members.

7.6 HOW TO USE FRIENDS

Example: /example/lec07/list

7.6 HOW TO USE FRIENDS

7.6 HOW TO USE FRIENDS

- Friends have the following properties:

7.6 HOW TO USE FRIENDS

- Friends have the following properties:
- Friends of a class can directly access all its members.

7.6 HOW TO USE FRIENDS

- Friends have the following properties:
- Friends of a class can directly access all its members.
- Friend declarations must be placed inside the class, but it doesn't matter which segment they are in.

7.6 HOW TO USE FRIENDS

- Friends have the following properties:
- Friends of a class can directly access all its members.
- Friend declarations must be placed inside the class, but it doesn't matter which segment they are in.
- Friendship is not symmetric, meaning if X is a friend of Y, Y may not necessarily be a friend of X.

7.6 HOW TO USE FRIENDS

- Friends have the following properties:
- Friends of a class can directly access all its members.
- Friend declarations must be placed inside the class, but it doesn't matter which segment they are in.
- Friendship is not symmetric, meaning if X is a friend of Y, Y may not necessarily be a friend of X.
- Friendship is not transitive, meaning if X is a friend of Y and Y is a friend of Z, X may not necessarily be a friend of Z.

7.6 HOW TO USE FRIENDS

Good code style: avoid using friends as much as possible!

EXERCISES

- Implement the following methods for Vector2D in both **static** and **non-static** versions
  - sub
  - dot
  - cross
- Implement Matrix2D using vector, and implement the following methods:
  - add
  - scale (matrix and scalar multiplication)
  - mult (matrix and matrix multiplication)
- Implement the following methods for vector
  - Non-static member function: applyTransform(const Matrix2D&)
  - Static member function: mult(const Matrix2D&, const Vector2D&)