C++ how to programming

Rational类的设计与实现

《程序设计2024秋》

湖南大学 – 信息科学与工程学院

本节任务——

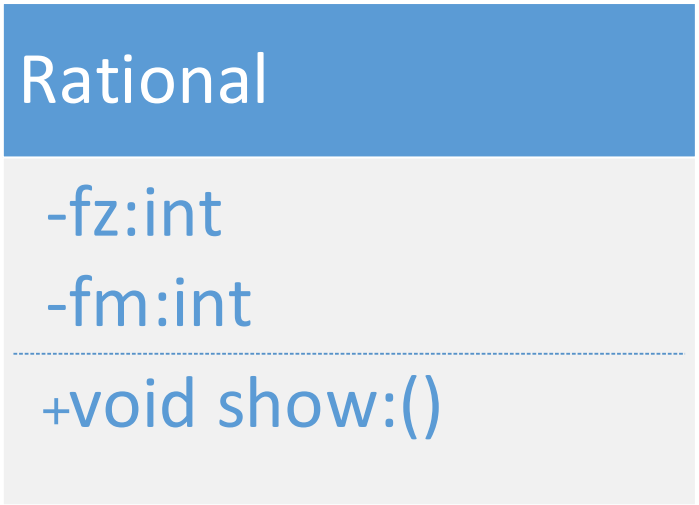
实现分数形式的有理数存储、输入输出和四则运算。

如：3/4 + 1/3 = 7/12 3/4 * 1/3 = 1/4

如何声明数据类型呢？int / char / int[2]

我的类型我定义 —— 构造数据类型 class

- 类是一种广义上的数据类型
- 关键字 class
- 类名：有意义的类型标识符（一般首字母大写）
- 封装两类成员：
  - 成员数据
  - 成员函数（操作成员数据）



```
class Rational{
  private:
    //数据成员声明：分子,分母;
    int fm;
    int fz;
  public:
    //成员函数声明：输出信息;
    void show();
};
```

```
int main(){
//声明一个有理数对象r
    Rational r;
//输出r的值
    r.show();
}
```

```
void Rational:: show() //成员函数定义
{   cout<<fz<<"/"<<fm<<endl;}
```

封装与数据隐藏

- 封装 Encapsulation
  - 将有关的数据和操作代码封装在一个对象中，形成一个基本单位，各个对象之间相对独立，互不干扰。
  - 将对象中某些部分对外隐蔽，即隐蔽其内部细节，只留下少量接口，有利于数据安全。(information hiding)
  - 类作用域
    - 类定义体
      - class Rational{};
    - 类的成员函数定义体
      - void Rational::show(){}
    - 类外
      - int main(){}

访问限定

- private、protected、public

	public	protected	private
类定义体	可见	可见	可见
成员函数定义体	可见	可见	可见
类作用域外	可见	子类可见 对象不可见	不可见

- 类域作用符 ::
- 类外定义成员函数，返回值类型之后，函数名前加类名::

小练习：为Rational类添加成员函数Set()，接受键盘输入的分子、分母值。

```
class Rational{
    private:
    //数据成员声明：分子,分母；
        int fm;
        int fz;
    public:
   //成员函数声明
        void show();
        void Set();
};
```

```
int main(){
//声明一个有理数对象r
   Rational r;
//调用Set()函数
   r.Set();
//输出r的值
   r.show();
}
```

```
void  Rational::show() //成员函数定义
{  cout<<fz<<"/"<<fm<<endl;}
void Rational::Set() //成员函数定义
{  cin >> fz >> fm;}
```

要对有理数的分子和分母进行初始化

```
class Rational{
   public:
    //增加成员函数：初始化数据成员constructor
    Rational(); //构造函数，函数名即类名，无返回值类型声明
};
```

```
Rational::Rational()
{  fz = 0; fm = 1;}
```

```
int main(){
//声明对象时自动调用构造函数
Rational r;
//r的值等于初始值0/1
r.show();
}
```

支持多种初始化方式，
比如b =1/2?
c = b?

—— 构造函数重载

要对有理数的分子和分母进行初始化

```
class Rational{
   public:
    //增加重载的构造函数
     Rational(int x , int y);
};
```

```
Rational::Rational(int x,int y)
{       fz = x;
        fm = y;}
```

```
int main(){
//声明对象时自动调用构造函数
Rational b(1,2);
//b的值等于初始值1/2
b.show();    }
```

函数重载：同名不同参

- 多个函数取相同的函数名
- 形参的个数或者形参的类型至少有一个不同
- 编译系统会根据实参的类型及个数，最佳匹配调用哪一个同名函数，即函数重载。
- 重载可以统一调用形式
- 重载函数的参数必须不同
- 返回值类型***不能区分***两个重载函数
- 重载函数必须功能相同或类似

要对有理数的分子和分母进行初始化

```
class Rational{
   public:
    //增加重载的构造函数，copy constructor
    Rational(const Rational& r);
};
```

```
Rational::Rational(const Rational& r)
{       fz = r.fz;
        fm = r.fm;}
```

```
int main(){
//声明对象时自动调用构造函数
Rational c(b);
//c的值等于初始值1/2
c.show();    }
```

构造函数 Constructor

- 函数名和类名相同，无返回值类型说明
- 功能：初始化数据成员
- 对象被创建时，系统自动调用
- 允许重载，以支持不同的对象初始化方式
- 复制构造函数，以该类型对象的常引用&为形参
  - 用已有对象初始化新对象
  - 形参为对象
  - 函数返回值为对象

……

JUST AI/DEBUG !

复制构造函数copy constructor

C++中的复制构造函数，在哪些情况下被调用

在C++中，复制构造函数（copy constructor）是一种特殊的构造函数，用于创建一个新对象并将其初始化为另一个同类型对象的副本。它在多种情况下会被调用。以下是一些常见的调用场景：

1. **显式复制**：

   当你显式地创建一个对象并用另一个同类型对象进行初始化时，复制构造函数会被调用。

```cpp
MyClass obj1;
MyClass obj2 = obj1;  // 复制构造函数被调用
```

2. **对象作为函数参数传递（按值传递）**：

   当对象作为函数参数按值传递时，调用复制构造函数来创建函数内部的参数副本。

```cpp
void func(MyClass obj) {
    // obj 是通过复制构造函数创建的 obj1 的副本
}

MyClass obj1;
func(obj1);  // 复制构造函数被调用
```

3. **函数返回对象（按值返回）**：

   当函数返回一个对象（按值返回）时，复制构造函数会被调用以将返回值复制到调用者的上下文中。

```cpp
MyClass func() {
    MyClass obj;
    return obj;  // 返回时调用复制构造函数
}

MyClass obj1 = func();  // 接收返回值时调用复制构造函数
```

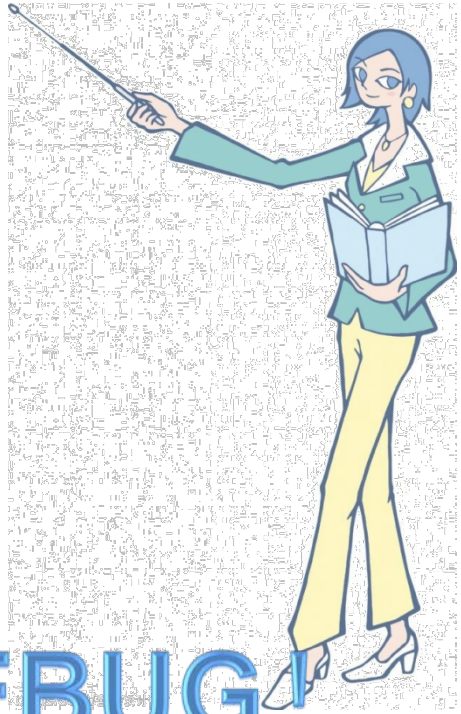
4. **对象数组初始化**：

   当使用另一个对象来初始化一个对象数组时，每个数组元素都会调用复制构造函数。

```cpp
MyClass obj1;
MyClass arr[3] = {obj1, obj1, obj1};  // 每个元素初始化时调用复制构造函数
```

析构函数 DeConstructor

- 函数名为~+类名，无返回值类型说明
- 功能：释放资源【类中包含指针成员，有动态内存分配】
- 不允许重载，有且仅有一个
- 对象声明周期结束时，系统自动调用
  - 超出作用域，如局部对象、临时对象
  - 使用delete操作符
  - 对象作为成员变量
  - 继承关系……



析构函数 DeConstructor

湖南大学 HUNAN UNIVERSITY

```cpp
#include <iostream>

class MyClass {
public:
    // 构造函数
    MyClass() {
        std::cout << "Constructor called"
    }

    // 析构函数
    ~MyClass() {
        std::cout << "Destructor called" <
    }
};
```

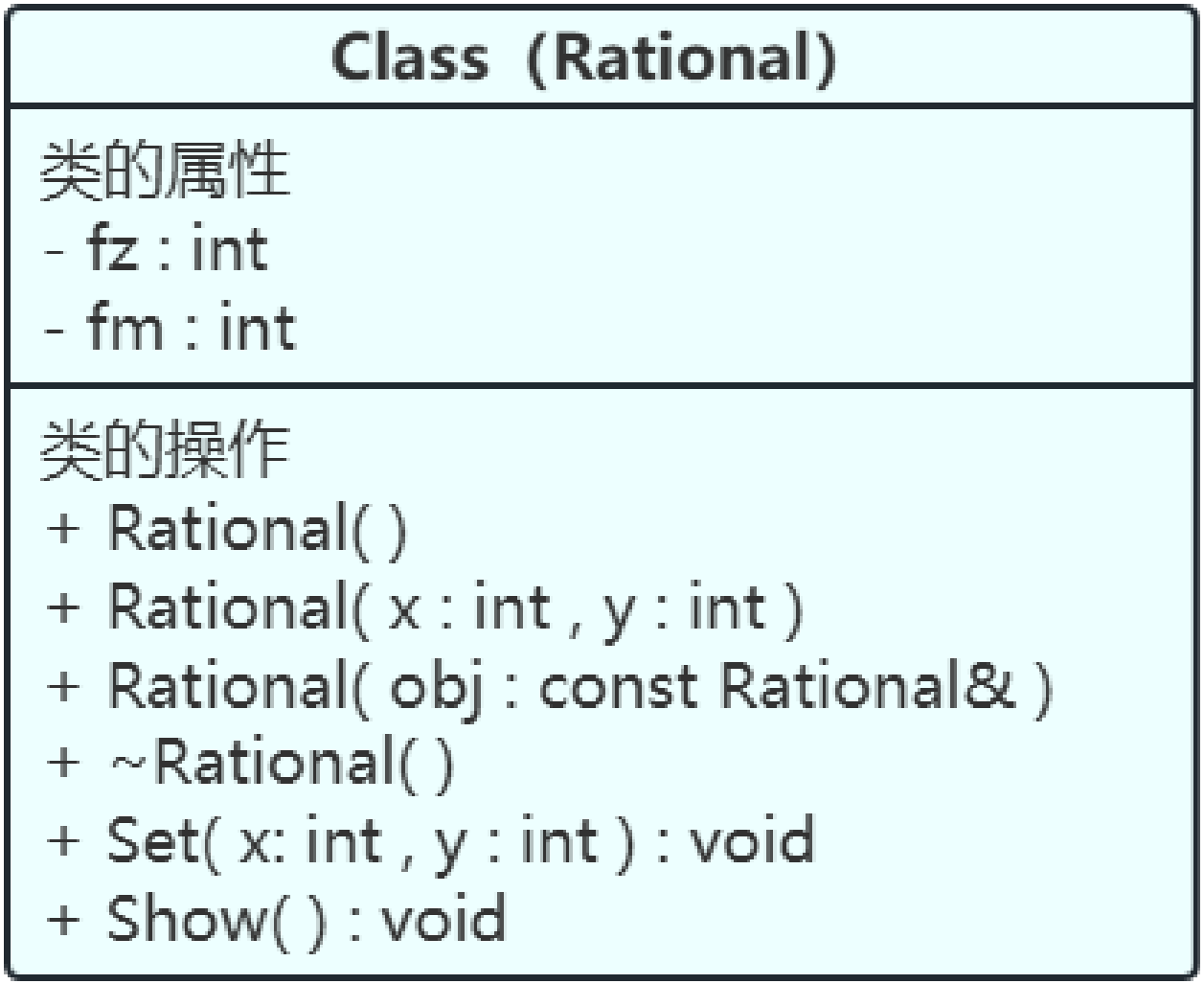
```cpp
int main() {
    // 情况一：对象生命周期自然结束
    {
        MyClass obj1; // 构造函数被调用
        // obj1 在这里可以使用
    } // 离开作用域，析构函数被调用

    // 情况二：使用delete操作符销毁动态分配对象
    MyClass* obj2 = new MyClass(); // 构造函数被调用
    // obj2 在这里可以使用
    delete obj2; // 析构函数被调用

    return 0;
}
```

```
Constructor called
Destructor called
Constructor called
Destructor called
```

UML类图 一键生成 代码



要对有理数进行乘法运算

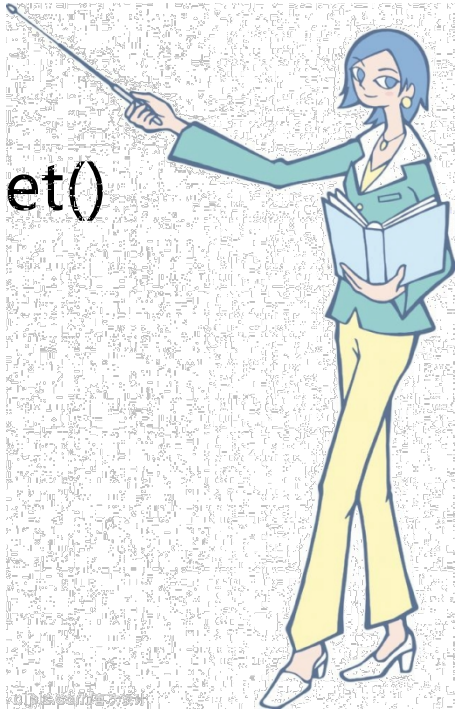
```
class Rational{
  public:
   //增加成员函数：相乘
 Rational mul (Rational);
 };
```

```
int main(){
   Rational a(1,4);
   Rational b(1,3);
   Rational c;
  //调用成员方法mul()
   c = a.mul(b);
}
```

```
Rational Rational::mul(Rational rObj)
{
   Rational result;
   result.fm = (this->fm) * (rObj.fm);
   result.fz = (this->fz) * (rObj.fz);
   return result;
}
```

成员函数

- 在类外访问public方法，通过对象名和.运算符
- 功能1：构造函数/析构函数
- 功能2：访问私有数据成员的接口，Set()/Get()
- 功能3：其他功能函数
- 成员函数可自由访问数据成员
  - 数据成员是声明在类中的全局变量
- 成员函数的第一个形参是隐含的this指针
  - this指针指代当前参与运算的对象（左对象）
  - ->运算符，访问类成员



第二种方式：要对有理数进行乘法运算

```
class Rational{
  public:
   //增加友元函数：相乘
friend Rational  mult(Rational,Rational);
};
```

```
int main(){
   Rational a(1,4);
   Rational b(1,3);
   Rational c;
  //调用友元方法mult()
   c = mult(a,b);
}
```

```
Rational mult(Rational lObj,Rational rObj)
{
    Rational result;
    result.fm = (lObj.fm) * (rObj.fm);
    result.fz = (lObj.fz) * (rObj.fz);
    return result;
}
```

友元函数

- 在函数声明的返回值类型之前，显式添加friend关键字
- 参与操作的每个对象都需声明为形参
- 在友元函数内部，可以自由访问类的私有成员
- 不是成员函数
  - 没有访问限定控制，没有this指针
  - 在类外定义友元函数，不用friend，不添加类名::
- 直接通过函数名来调用友元函数

基于类的编程三步骤

- 类的定义
  - class{ 成员数据+成员函数 }; —— 封装 Encapsulation
  - public、private、protected、friend
- 成员/友元函数的实现
  - 类外定义，返回值类型和成员函数名之间加类名::
- 创建对象并测试对象的功能
  - 类名就是类型标识符
  - 调用方法—— 对象名. 公有成员名
- 类是实现面向对象程序设计的基础

课后练习：多文件工程

- 头文件：定义完整的Ratioanl类 —— rational.h
  - #pragma once 预防重复编译的预处理命令
- 源文件：成员/友元函数的实现—— rational.cpp
  - #include “rational.h” 包含类声明的头文件
- 测试文件：创建对象并测试对象功能 —— main.cpp
  - #include “rational.h” 包含类声明的头文件
  - 类名就是类型标识符
  - 公有成员调用方法—— 对象名. 公有成员名
  - 友元函数调用方法—— 函数名(实参对象列表)