# 封装

面向对象中的封装，不但封装了数据，还封装了行为。

# 访问属性

公有 私有 保护

体现了数据隐藏的思想。

# 接口

指类的公有的函数成员，通过他们可以在类的外部间接地访问类的私有成员，相当于在类的私有成员和外部建立了一个桥梁。接口是一个形象的说法，把本来隐藏的部分对外开了一个口子。

# 构造函数

C语言对普通变量进行初始化。

int a = 4;

如何对C++中的对象进行初始化？

难点在于一个对象里面有很多数据。

int a(4); //int a = 4;

构造函数的目的：用来对对象进行初始化的。

C语言的函数调用

G(){

}

F(){

G();

}

特点

1. 是一个成员函数。
2. 在创建对象的时候，由系统自动调用的。
3. 函数名和类名一样，编译器才知道他是构造函数
4. 构造函数无返回值（不写void）
5. 访问属性应该设置为 公有
6. 在创建对象的时候，编译器必须调用对应构造函数。
7. 构造函数可以重载，支持以多种方式去初始化对象。
8. 如果没有写构造函数，编译器会产生一个无参的构造函数，这个函数里面没有代码。

被称为默认构造函数。如果写了构造函数，默认构造函数就没有了。

注意：当构造函数具有默认形参值的时候，不要和其他的构造函数在调用形式上产生冲突。

#include <iostream>

using namespace std;

class Student{

public:

int id;

int age;

int score;

void eat();

// void eat(){

// cout<<"去吃饭"<<endl;

// }

void setMoney(int m){

money = m;

}

Student(int i=10,int a=20,int s=80,int m=100){

id = i;

age = a;

score = s;

money = m;

}

Student(){

id = 1;

age = 20;

score = 80;

money = 100;

}

private:

int money;

};

void Student::eat(){

if (money>100)

cout<<id<<" 去馆子"<<endl;

else

cout<<id<<" 去食堂"<<endl;

}

int main(){

//struct Student s;

//Student s(1,20,80,120);

//s.id = 7;

//s.age = 20;

//cout<<s.id<<" "<<s.age<<endl;

//s.money = 120;

//s.setMoney(120);

//s.eat();

//Student.eat();

//s.money = 100;

Student s1;

}

# 析构函数

析构函数的目的：当对象消亡的时候，进行一些清理的工作。

比如 大四的学生

析构函数的特点：

1. 在对象消亡的时候被自动调用的，所以适合在析构函数中做清理工作
2. 函数形式 ~类名（），不写void
3. 析构函数应该是公有的，不能重载
4. 通常根据业务的需要，在构造函数里申请资源，在析构函数中释放资源。

#include <iostream>

using namespace std;

class Student{

public:

int id;

int age;

int score;

int \* p;

void eat();

// void eat(){

// cout<<"去吃饭"<<endl;

// }

void setMoney(int m){

money = m;

}

Student(int i,int a=20,int s=80,int m=100){

id = i;

age = a;

score = s;

money = m;

}

Student(){

id = 1;

age = 20;

score = 80;

money = 100;

cout<<"我来了"<<endl;

p = new int[5];

}

~Student(){

cout<<"我死了"<<endl;

delete[] p;

}

private:

int money;

};

void Student::eat(){

if (money>100)

cout<<id<<" 去馆子"<<endl;

else

cout<<id<<" 去食堂"<<endl;

}

int main(){

//struct Student s;

//Student s(1,20,80,120);

//s.id = 7;

//s.age = 20;

//cout<<s.id<<" "<<s.age<<endl;

//s.money = 120;

//s.setMoney(120);

//s.eat();

//Student.eat();

//s.money = 100;

Student s1;

}

案例 圆形类的应用。

//4\_3.cpp

#include <iostream>

using namespace std;

const float PI = 3.141593; //给出p的值

const float FENCE\_PRICE = 35; //栅栏的单价

const float CONCRETE\_PRICE = 20;//过道水泥单价

class Circle { //声明定义类Circle 及其数据和方法

public: //外部接口

Circle(float r); //构造函数

float circumference(); //计算圆的周长

float area(); //计算圆的面积

private: //私有数据成员

float radius; //圆半径

};

//类的实现

//构造函数初始化数据成员radius

Circle::Circle(float r) {

radius = r;

}

//计算圆的周长

float Circle::circumference() {

return 2 \* PI \* radius;

}

//计算圆的面积

float Circle::area() {

return PI \* radius \* radius;

}

//主函数实现

int main () {

float radius;

cout << "Enter the radius of the pool: "; // 提示用户输入半径

cin >> radius;

Circle pool(radius); //游泳池边界

Circle poolRim(radius + 3); //栅栏

//计算栅栏造价并输出

float fenceCost = poolRim.circumference() \* FENCE\_PRICE;

cout << "Fencing Cost is $" << fenceCost << endl;

//计算过道造价并输出

float concreteCost = (poolRim.area() - pool.area()) \* CONCRETE\_PRICE;

cout << "Concrete Cost is $" << concreteCost << endl;

return 0;

}

# 复制构造函数

在定义一个新的变量的时候，用一个已有的变量对他进行初始化。

int b(a);

对于对象，如何在定义一个新的对象的时候，用一个已有的对象对他进行初始化？

Student s1(s);

通常，应该用同类的对象来初始化新的对象。

对于对象，如何在定义一个新的对象的时候，用一个已有的对象对他进行初始化，并且在初始化的过程中进行必要控制？有选择的利用旧对象的数据对新对象进行初始化。

复制构造函数的作用：

1）在使用复制构造的方式创建对象的时候被自动调用的。Student s1(s);

2）通过自定义复制构造函数，我们可以控制复制的过程。

3）名字：类名。参数：必须是同类对象的引用。

4）如果没有自定义复制构造函数，系统会自动生成一个默认的复制构造函数，实现的是数据的完全复制。

关于创建对象时对构造函数的调用：

只要创建了对象，就必然会去调用对应构造函数或者调用复制构造函数。

#include <iostream>

using namespace std;

class Student{

public:

int id;

int age;

int score;

int \* p;

void eat();

// void eat(){

// cout<<"去吃饭"<<endl;

// }

void setMoney(int m){

money = m;

}

Student(int i,int a=20,int s=80,int m=100){

id = i;

age = a;

score = s;

money = m;

}

Student(){

id = 1;

age = 20;

score = 80;

money = 100;

cout<<"我来了"<<endl;

p = new int[5];

}

// Student(Student &s){

// id = s.id + 1;

// age = s.age;

// score = s.score;

// money = s.money;

// }

~Student(){

cout<<"我死了"<<endl;

delete[] p;

}

private:

int money;

};

void Student::eat(){

if (money>100)

cout<<id<<" 去馆子"<<endl;

else

cout<<id<<" 去食堂"<<endl;

}

int main(){

//struct Student s;

Student s(1,20,80,120);

//s.id = 7;

//s.age = 20;

//cout<<s.id<<" "<<s.age<<endl;

//s.money = 120;

//s.setMoney(120);

//s.eat();

//Student.eat();

//s.money = 100;

Student s1(s);

cout<<s1.id<<endl;

}