221900180 田永铭 作业一

概念题：

1.

开发效率：面向对象编程使得我们能够在软件开发时，根据需要将现实世界的事物抽象成类，并进行封装，提供接口，从而能够进行程序员之间的代码合作和交流。这样可以将原本繁杂的大项目分解，通过多人合作开发，大大提高效率。更好地，面向对象编程使得代码能够多线程运行，这种方式更贴近人类的日常思维，能够提高程序的开发效率。同时继承、多态性也能提高开发效率。

软件质量：面向对象编程风格的代码在编写上有一定要求，因为它要提供对外交流的接口，因此对于优秀的程序员，在面向对象编程时代码风格更好更优秀，这使得软件更好理解，更容易扩展和维护；这种编程模式下，软件可复用，软件可维护性强，软件模型的自然度好，因为语言更人性化。总体上，软件质量更高了。

2.

对象：由数据以及能对其实施的操作所构成的封装体。对象构成了面向对象程序的基本计算单位。

类：类描述了对象的特征（数据和操作特征），实现抽象，用以创建对象。

关系：由类才有对象，对象是用类来创建的。而对象的特征则由相应的类来描述。

3.

a,b,d,f2 只能在本类和友元的代码中访问;

e,f3 只能在本类、派生类和友元的代码中访问;

c,f1,f4 访问不受限制。

4.

对于常量和引用数据成员，C++旧版本不能在说明它们时初始化，也不能采用赋值操作在构造函数中对它们初始化。因此原代码仅有 int x; 与 x = 0; 这两行正确，其他都错误。

解决方法：可以在构造函数的函数头和函数体之间加入一个成员初始化表来对常量和引用数据成员进行初始化。例如：

```
class A
 {   int x;
        const int y;
        int& z;
   public:
        A(): z(x),y(1)
        {
           x = 0;
        }
   };
```

5.

如果对象创建后，自己又额外申请了资源（如：额外申请了内存空间），则可以自定义析构函数来归还它们。例如对象自己申请了个 str = new char[len+1];此时就要写个析构函数

归还空间。

析构函数能够归还大部分的资源，如很多申请的空间等等，而不能归还直接管理另一块内存资源的类申请的空间，或如果成员变量含有指针，并且指针指向一块我们正使用的空间，编译器生成的析构函数就不能归还这种资源。

编程题

1.

```
class Account
{
	private:
	double money;	//钱数
	char name[15];	//名字
	char acc[105];	//账号

	public:
	Account():money(…),name(…),acc(…);	//初始化
	void show();	//展示姓名、账号和存款
	void save(int sum);	//存款
	void retreat(int sum);	//取款
}
```

2.

```
class A
{
	private:
	int x,y;

	public:
	void f();
	void g(int i)
	{
		x = i;
		f();
	}
};

A a,b;
a.f();
a.g(1);
b.f();
b.g(2);
```

3.

```
struct ListNode {
    int val;
    ListNode * next;
 ListNode(int val=-1, ListNode * next=nullptr): val(val), next(next) {}
};

ListNode * reverseList(ListNode * head)
{
    if(head == NULL) return head;              //空指针就直接返回
    else
    {
        ListNode *p1 = head;                          //利用三个指针同步向后移动
        ListNode *p2 = head->next;                    //逐步实现链表翻转
        ListNode *p3 = NULL;
        p1->next = NULL;              //让反转后的链表尾巴指向空指针
        while(p2 != NULL)
        {
            p3 = p2->next;
            p2->next = p1;
            p1 = p2;
            p2 = p3;
        }
        head = p1;                                    //返回链表头
        return head;
    }
}
```