Lecture 9: Abstraction III
Methods

Introduction

A *method*, also known as a class function, is the last type of class member. It implicitly consists of the this pointer, allowing it to operate on the correct class object. Additionally, a class can grant access to its members to non-members deemed as friends.

Methods

There are three types of methods:

1. **Instance Method** - has the this pointer, providing read/write access to the object's fields and permitting all method invocations.
2. **Constant Method** - has a constant this pointer, providing readonly access to the object's fields and permitting only constant and static method invocations.
3. **Static Method** - does not have the this pointer; provides read/write to static fields and permits only static method invocations.

All methods are accessible through class objects using the dot operator. Static methods can also be accessed without an object using the scope resolution operator. Methods can be defined within or outside the class. If defined outside, their declaration must be included in the class.

Instance and constant methods are considered distinct (overloaded) even with the same function signatures. Non-constant methods and objects invoke the instance method version if both are provided. A static method cannot be constant since static methods do not have access to the this pointer.

Their syntaxes are:

- **Instance Method**: *return-type identifier*(*parameter-list*) {*body*}

  Invoked by non-constant objects and other instance methods.
- **Constant Method**: *return-type identifier*(*parameter-list*) const {*body*}

  Invoked by any object and method.
- **Static Method**: static *return-type identifier*(*parameter-list*) {*body*}

  Invoked by any object, method, or the class itself.

Example:

```
class O
{
  public:
  O() : a(0), b(0) {}
  O(int a,int b) : a(a), b(b) {}
  int a;
  int sum() {return (a + b + c);}
  int avg() {return sum() / 3;}
  int sum() const {return 2 * (a + b + c);}
  int prd() const {return (a * b * c);}
  int b;
  static void dub() {c *= 2;}
  static int c;
};
int O::c = 5;
```

In the above class, after the constructors, the first two, second two, and final methods are instance, constant, and static methods, respectively. Given this class following is possible.

```
int main()
{
  O a(2,7);
  const O b(1,3);
  std::cout << a.sum() << '-' << b.sum() << '\n'; //14-18
  std::cout << a.avg() << '\n'; //4
  std::cout << a.prd() << '-' << b.prd() << '\n'; //70-15
  O::dub();
  std::cout << a.sum() << '-' << b.sum() << '\n'; //19-14
  a.dub();
  std::cout << a.c << '-' << b.c << '-' << O::c << '\n';//20-20-20
  return 0;
}
```

toString() Method & sstream Library

Object-oriented programming languages typically have a method to represent objects as a string, known as the `toString()` method. It can be implemented as follows:

`string toString() const {`*body*`}`

To help define the method, a *ostringstream* object from the *sstream* library can be used. An ostringstream object builds a string using stream operations like the `cout` object. Once the string is completely built, the resultant string can be retrieved using the `str()` method.

Example:
For the class *O*

```
string toString() const
{
  stringstream out;
  out << "("<< a << ","<< b << ","<< c << ")";
  return out.str();
}
```

Friendship

A class can provide a function or another class access to its private and protected members using the *friend* statement. The syntax is:

1. `friend` *declaration*
2. `friend` *definition*

Using version 1, the friend function or class is defined after the class, without the '`friend`' keyword.

Ostream Operator To allow a class object to be displayed using an ostream object, the class must overload the ostream operator and make the operator a friend. Its syntax is

`friend std::ostream& operator<<(std::ostream&` *identifier*`,const` *class*`&` *identifer*`){`*body*`}`

Example:
If a class defines a `toString()` method, the overload ostream operator can be defined within the class as

```
friend ostream& operator<<(ostream& out,const class& obj)
{
  out << obj.toString();
  return out;
}
```