CS256

Chapter 14: More About Classes

Hanoi University of Science and Technology

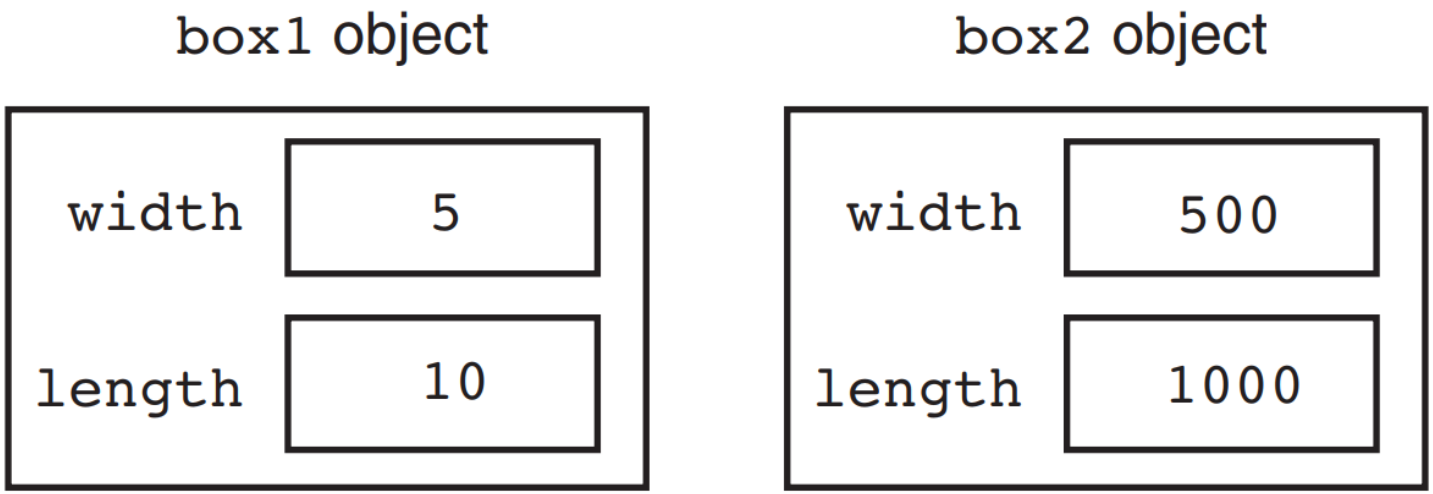
Slide contents are mainly based on the following documents:

- Tony Gaddis, Starting out with C++, Eighth Edition, Pearson, 2015
- Tony Gaddis, Power Points Slides:
  `https://www.umsl.edu/~antognolij/gaddis.html`

Instance and Static Members

```
Rectangle box1, box2;

// Set the width and length for box1.
box1.setWidth(5);
box1.setLength(10);

// Set the width and length for box2.
box2.setWidth(500);
box2.setLength(1000);
```

Instance and Static Members



Instance and Static Members

- **Instance variable:** a member variable in a class. Each object has its own copy.
- **Static variable:** one variable shared among all objects of a class.
- **Static member function:** can be used to access static member variables; can be called before any objects are defined.

Static Member Variable: Tree.h

```
// Tree class
class Tree {
private:
    static int objectCount; // Static member variable.
public:
    // Constructor
    Tree() { objectCount++; }

    // Accessor function for objectCount
    int getObjectCount() const { return objectCount; }
};

// Definition of the static member variable, written
// outside the class.
int Tree::objectCount = 0;
```

Static Member Variable: `Tree.h`

Program 14-1

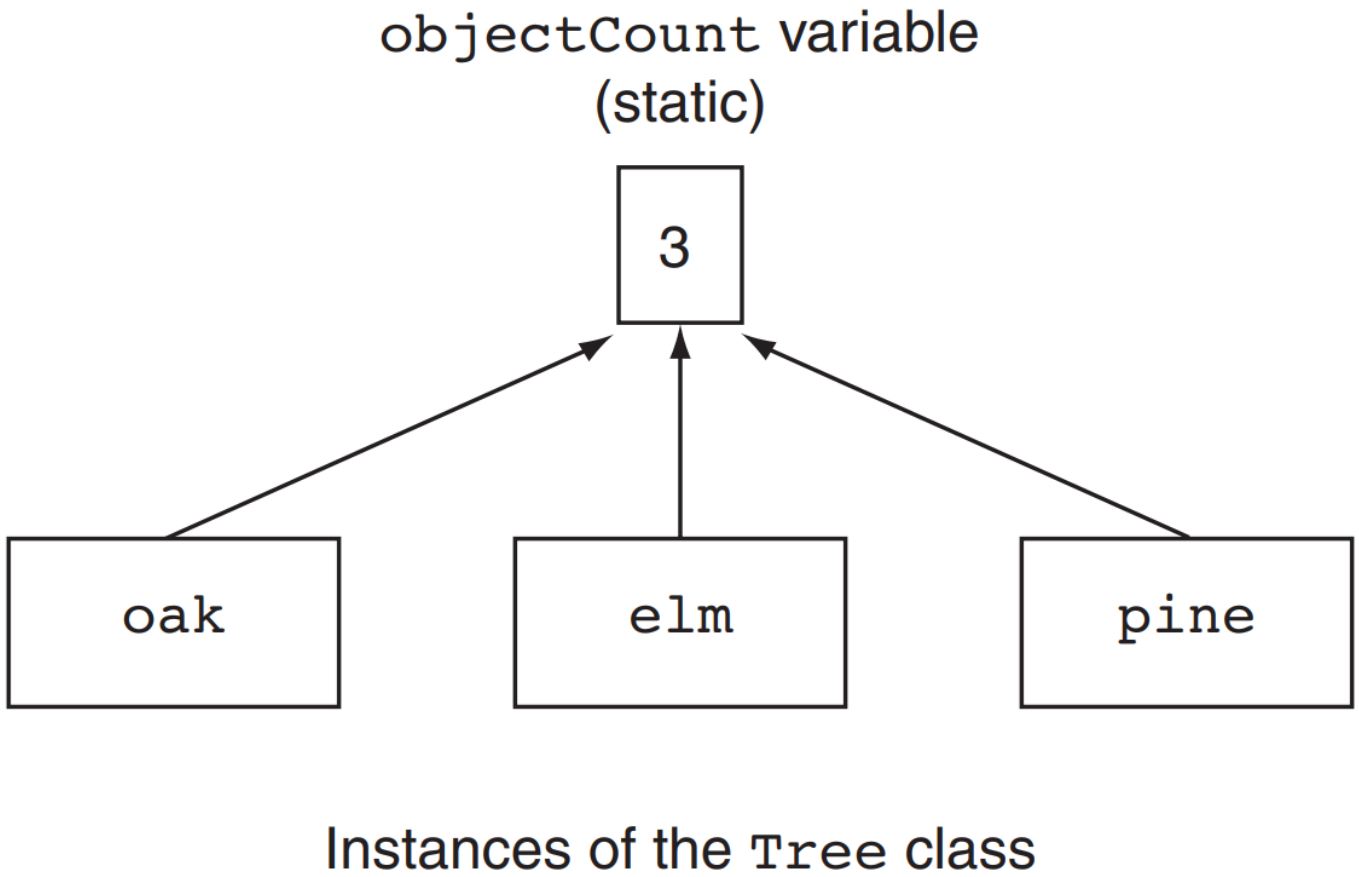
```
1  // This program demonstrates a static member variable.
2  #include <iostream>
3  #include "Tree.h"
4  using namespace std;
5
6  int main()
7  {
8       // Define three Tree objects.
9       Tree oak;
10      Tree elm;
11      Tree pine;
12
13      // Display the number of Tree objects we have.
14      cout << "We have " << pine.getObjectCount()
15           << " trees in our program!\n";
16      return 0;
17 }
```

Program Output

```
We have 3 trees in our program!
```

Static Member Variable: `Tree.h`

Three Instances of the `Tree` Class, But Only One `objectCount` Variable



Static Member Function

- Declared with `static` before return type:

```
static int getObjectCount() const
{
    return objectCount;
}
```

- Static member functions can only access static member data.
- Can be called independent of objects:

```
int num = Tree::getObjectCount();
```

Memberwise Assignment

- Can use = to assign one object to another, or to initialize an object with an object's data
- Copies member to member. e.g.,

  `instance2 = instance1;` means:

  copy all member values from `instance1` and assign to the corresponding member variables of `instance2`
- Use at initialization:

  ```
  Rectangle r2 = r1;
  ```

Memberwise Assignment

Program 14-5

```
1  // This program demonstrates memberwise assignment.
2  #include <iostream>
3  #include "Rectangle.h"
4  using namespace std;
5
6  int main()
7  {
8       // Define two Rectangle objects.
9       Rectangle box1(10.0, 10.0);   // width = 10.0, length = 10.0
10      Rectangle box2 (20.0, 20.0);  // width = 20.0, length = 20.0
11
12      // Display each object's width and length.
13      cout << "box1's width and length: " << box1.getWidth()
14           << " " << box1.getLength() << endl;
15      cout << "box2's width and length: " << box2.getWidth()
16           << " " << box2.getLength() << endl << endl;
17
18      // Assign the members of box1 to box2.
19      box2 = box1;
20
21      // Display each object's width and length again.
22      cout << "box1's width and length: " << box1.getWidth()
23           << " " << box1.getLength() << endl;
24      cout << "box2's width and length: " << box2.getWidth()
25           << " " << box2.getLength() << endl;
26
27      return 0;
28 }
```

Memberwise Assignment

Program Output

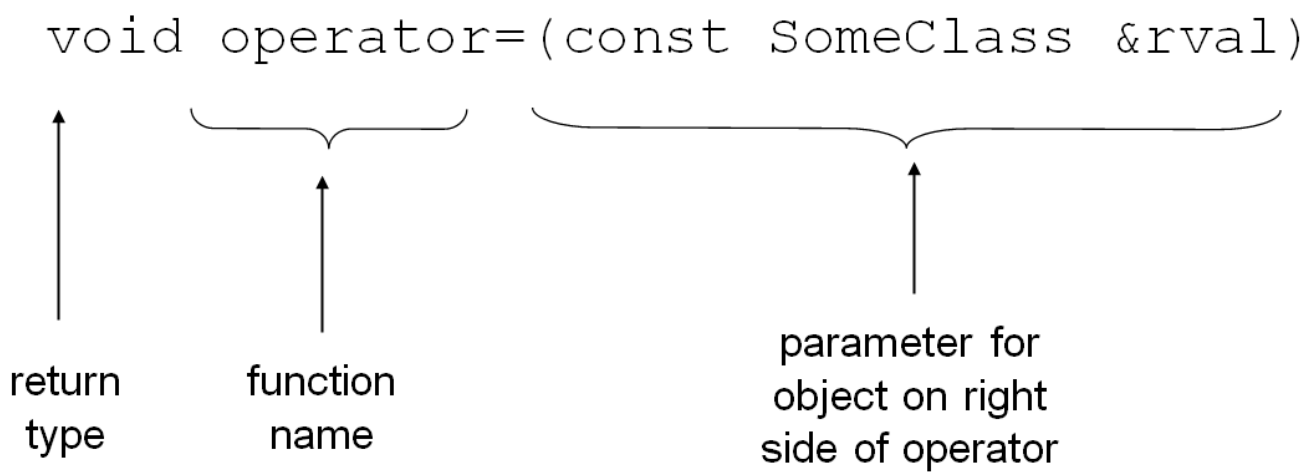
```
box1's width and length: 10 10
box2's width and length: 20 20

box1's width and length: 10 10
box2's width and length: 10 10
```

Operator Overloading

- Operators such as =, +, and others can be redefined when used with objects of a class.
- The name of the function for the overloaded operator is `operator` followed by the operator symbol, e.g.:
  `operator+`: to overload the + operator
  `operator=`: to overload the = operator
- Prototype goes in the declaration of the class that is overloading it.

Operator Overloading

- Prototype: `void operator=(const SomeClass &rval)`



Invoking an Overloaded Operator

- As a member function: `object1.operator=(object2);`
- Conventional manner: `object1 = object2;`

The this Pointer

- this: predefined pointer available to a class's member functions
- Always points to the instance (object) of the class whose function is being called
- Is passed as a hidden argument to all non-static member functions
- Can be used to access members that may be hidden by parameters with same name

```
class SomeClass {
private:
    int num;
public:
    void setNum(int num) {
        this->num = num;
    }
};
```

Notes on Overloaded Operators

- Can change meaning of an operator.
- Cannot change the number of operands of the operator.
- Only certain operators can be overloaded.
- **Cannot** overload: `?:`, `.`, `.*`, `::`, `sizeof`.
- `++`, `--` operators overloaded differently for prefix vs. postfix notation.
- Overloaded relational operators should return a bool value.
- Overloaded stream operators >>, << must return reference to istream/ostream objects.

Object Conversion

- Type of an object can be converted to another type
- Automatically done for built-in data types
- Must write an operator function to perform conversion
- To convert an `FeetInches` object to an int:

```
FeetInches::operator int() {return feet;}
```

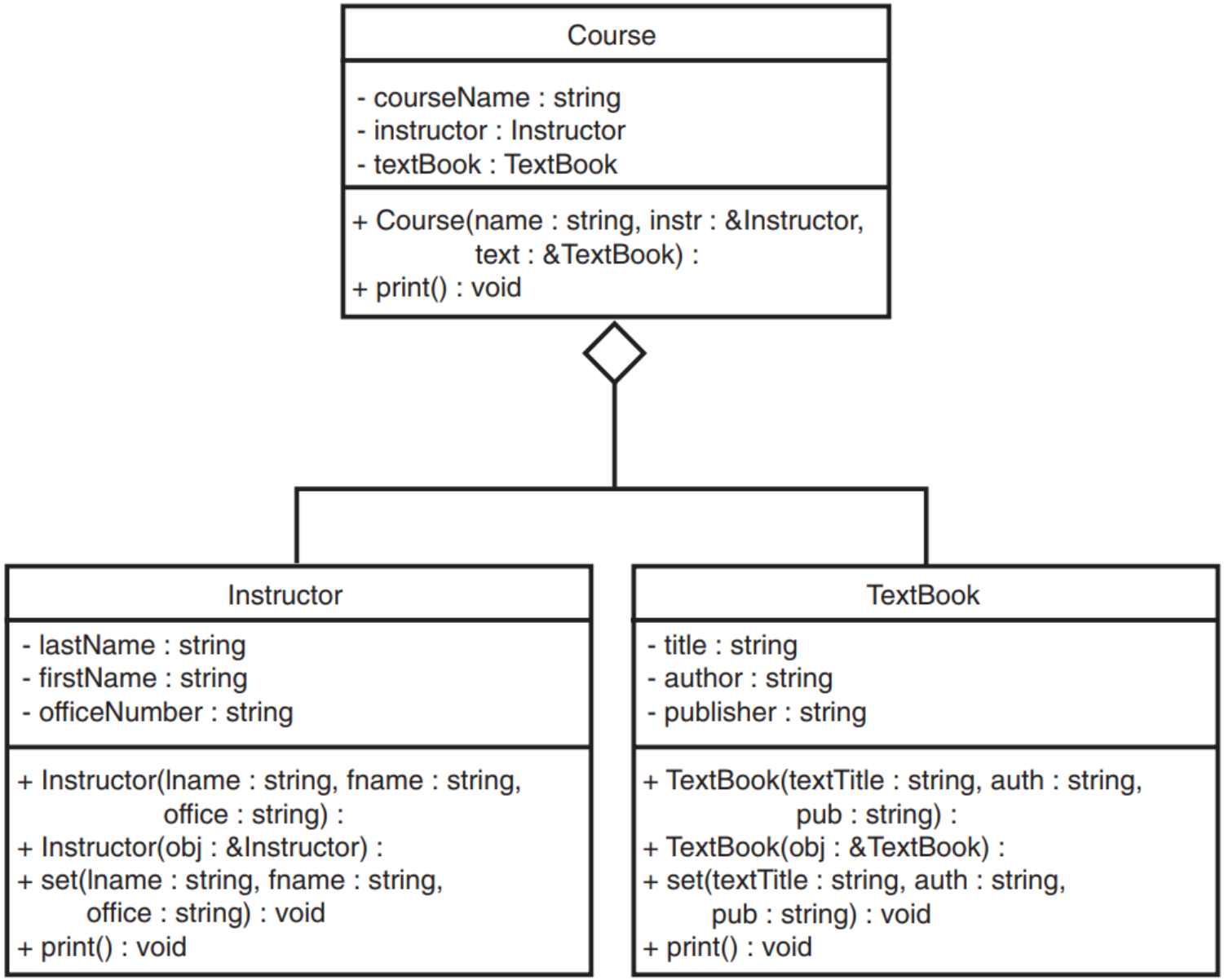
- Assuming `distance` is a FeetInches object, allows statements like:

```
int d = distance;
```

Aggregation

```
class StudentInfo {
private:
    string firstName, LastName;
    string address, city, state, zip;
};

class Student {
private:
    StudentInfo personalData;
};
```

Aggregation



Friends of Classes

- **Friend:** a function or class that is not a member of a class, but has access to private members of the class.
- A friend function can be a stand-alone function or a member function of another class.
- It is declared a friend of a class with the `friend` keyword in the function prototype.

Friends of Classes

- Stand-alone function: `friend void setAVal(intVal&, int);`
- Member function of another class:

  ```
  friend void SomeClass::setNum(int num);
  ```