### Chapter 10

### Classes



### **OBJECTIVES**

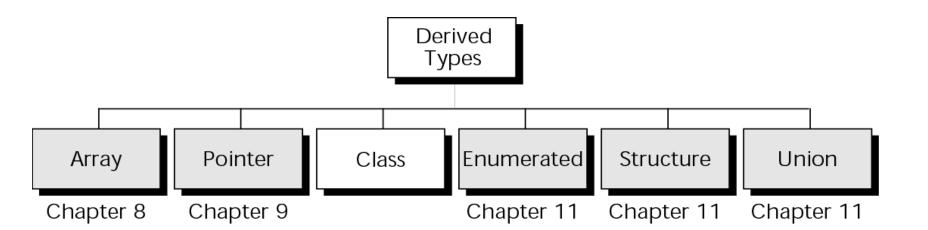
### After studying this chapter you will be able to:

- **■** Write programs that use simple class structures.
- Declare class types.
- Understand and use the private, protected, and public access.
- Define a function as a member of a class using the scope resolution operator.
- Access class members using the member operator.
- **□** Write class constructors and destructors.
- Understand and explain manager, mutator, and accessor functions.
- ☐ Use the Unified Modeling Language (UML) to design and document classes.



#### Figure 10-1 Derived types



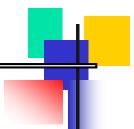


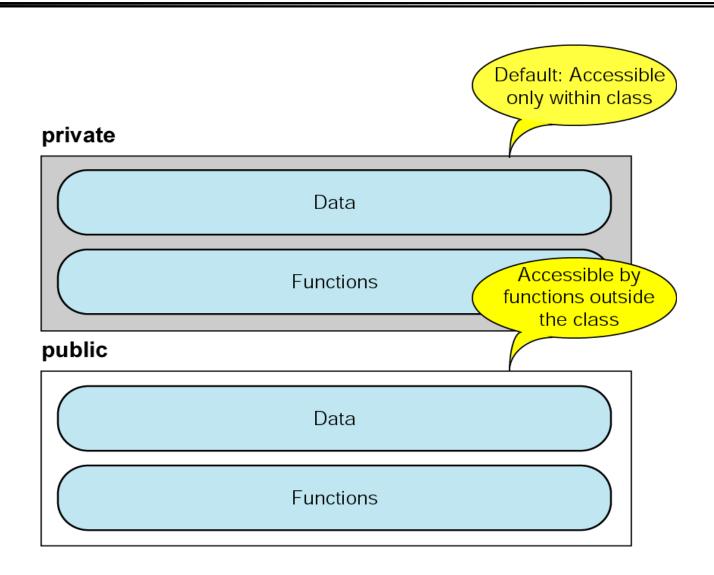
10.1

# BASIC CLASS CONCEPTS



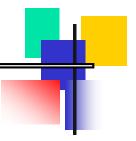
### Figure 10-2 Class access specifiers

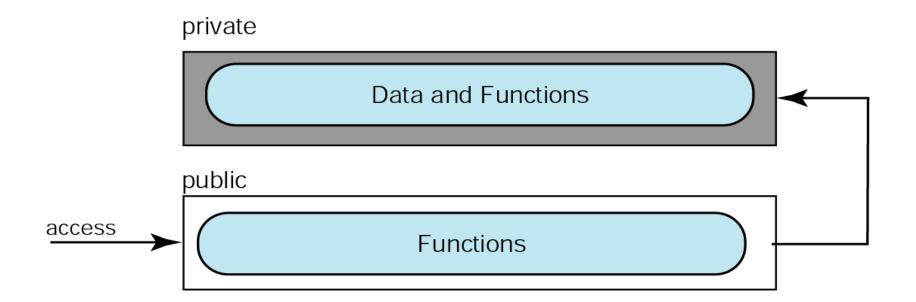






### Figure 10-3 Accessing data in a class



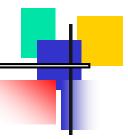




We recommend that class identifiers start with an uppercase letter to ensure uniqueness and to help identify the type as a class.



### Figure 10-4 Scope resolution operator



```
class Fraction
{
...
void print() const;

Scope Resolution

Void Fraction:: print () const

{

Class Declaration

Class Declaration

Scope Resolution

Void Fraction:: print () const

{

Function
Name

} // Fraction print
```

**Function Definition** 

10.2

# CLASS OBJECTS



Use the scope resolution operator to refer to a specific class type and use the member operator to refer to data or functions within an instantiated class object.



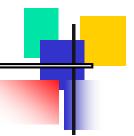
### Figure 10-5 Class objects

```
class Fraction
  private:
   int numerator;
                                    void Fraction :: store
   int denominator:
                                        (int numer,
 public:
                                         int
                                              denom)
   void store (int numer,
                                        numerator = numer;
                int denom);
                                   void Fraction :: print() const
                                        cout << numerator
   void print () const;
                                             << "/"
                                             << denominator:
  ; // class Fraction
                                        return;
(a) Class Declaration
                                    } // Fraction print
                                   (b) Function Definitions
 int main ()
                                            numerator denominator
                                                                  fr1
    Fraction
               fr1;
    Fraction
               fr2;
                                            numerator denominator
                                                                  fr2
 } // main
```

(c) Class Instantiation



### Figure 10-6 The this pointer



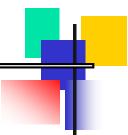


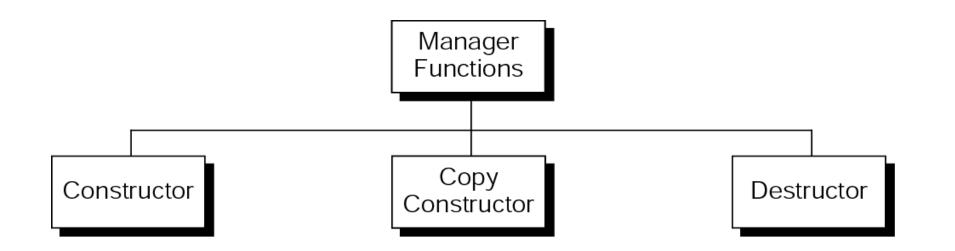
10.3

### MANAGER FUNCTIONS



### Figure 10-7 Manager Functions





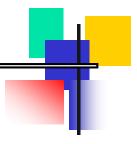
A constructor is called every time an object is instantiated.

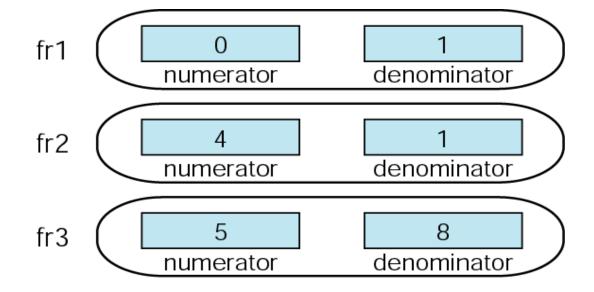


The name of a constructor is the same as the name of the class and it may not have a return type.



### Figure 10-8 Fractions created by Program 10-7







A class must have at least one constructor, either defined by the program or by the compiler.



The default constructor is called whenever an object is crated without passing any arguments.



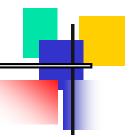
If we define any type of constructor, we must also define a default constructor if it is needed.

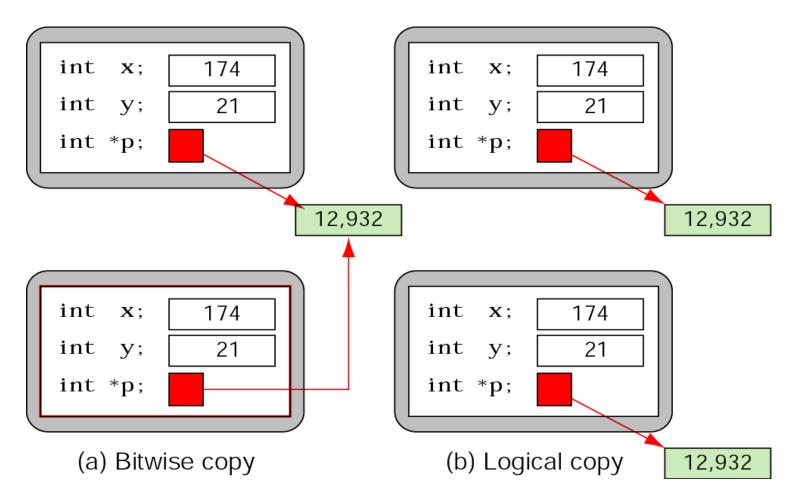


There is only one default constructor; it cannot be overloaded.



### Figure 10-9 Bitwise and logical copies





There is always one copy constructor available in a class.



Always pass objects to a function by reference; when the object cannot be changed, pass it as a constant.



Always return objects by value.



A class can have one, and only one, destructor.



10.4

## MUTATORS AND ACCESSORS

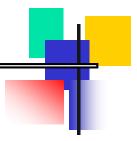


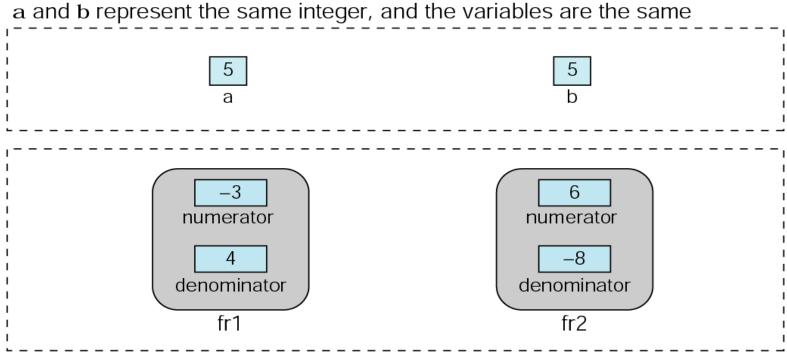
10.5

## CLASS INVARIANTS



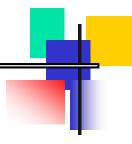
#### Figure 10-10 Class invariants





 ${\tt fr1}$  and  ${\tt fr2}$  represents the same fraction, but the objects are not the same

### Figure 10-11 Greatest common divisor



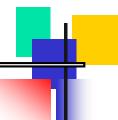
$$\gcd(x, y) = \begin{bmatrix} x & & \text{if } y = 0 \\ \gcd(y, x \mod y) & \text{otherwise} \end{bmatrix}$$

10.6

# COMPLEX CLASS FUNCTIONS



### Figure 10-12 Add two fractions



```
fr2
                 Parameter
void Fraction :: addTo
       (const Fraction& fr2)
                                          fr1
                                                      denominator
                                            numerator
                            this
  // Fraction addTo
        fr1.addTo (fr2);
```



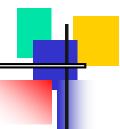
Functions that are members of a class can only be called through objects of that class. They cannot be called independently.



Friend functions are not members of a class, but are associated with it.



### Figure 10-13 Add and return fraction value



```
fr1
                     fr2
            Parameters
Fraction add
       (const Fraction& fr1,
        const Fraction& fr2)
{
    return Fraction(...);
    // Fraction addFr
           Return Value
    fr3 = add (fr1, fr2);
```

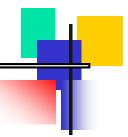


10.7

# PROGRAMMING APPLICATIONS



#### Figure 10-14 Time class design



#### **Time**

- hours
- minutes
- seconds
- convertToSeconds
- + increment
- + print
- + diff

#### Access:

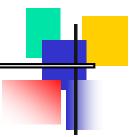
- private
- + public

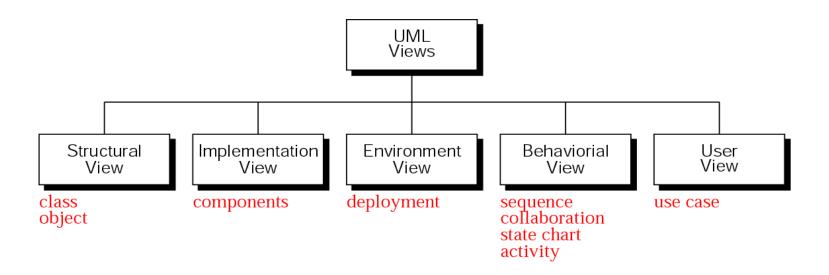


# SOFTWARE ENGINEERING MID PROGRAMMING STYLE



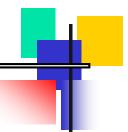
#### Figure 10-15 UML views







#### Figure 10-16 Class diagrams



Name only

Fraction

Name and Attributes

#### **Fraction**

numerator denominator Name and Operations

#### **Fraction**

store()
print()
add()
sub()
mul()

div()

Name, Attributes, and Operations

#### **Fraction**

numerator denominator

store() print() add() sub() mul() div()



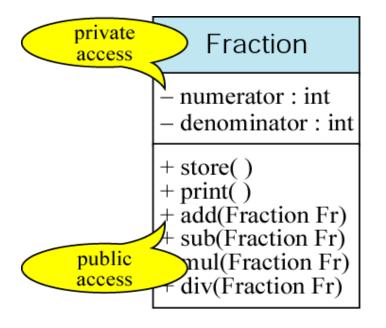
Note:

# The name compartment is mandatory in a class diagram.



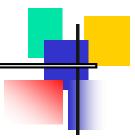
#### Figure 10-17 Expanded class diagram







#### Figure 10-18 An abstract class



**Employee** 

#### **Employee**

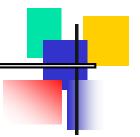
name address

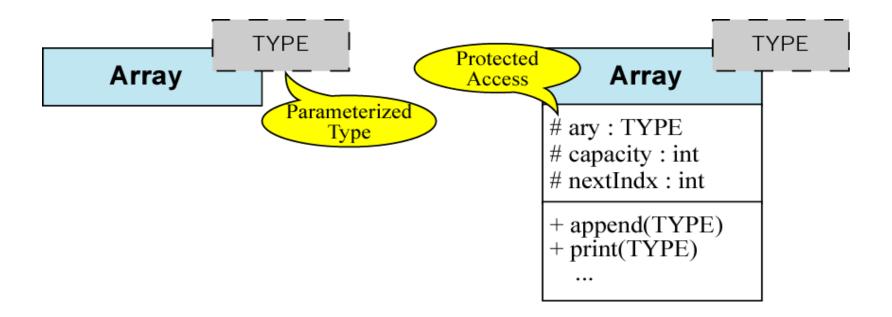
•••

```
store()
print()
hire()
promote()
adjSalary()
```



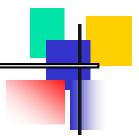
#### Figure 10-19 Class template diagrams

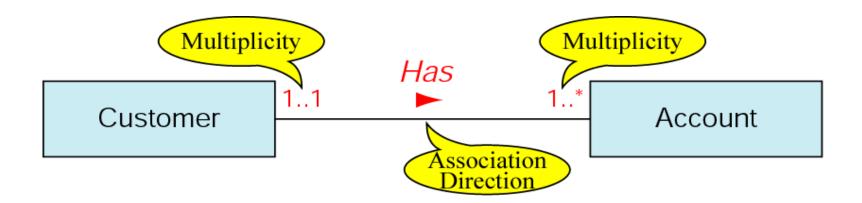






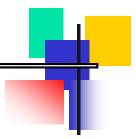
### Figure 10-20 Class association

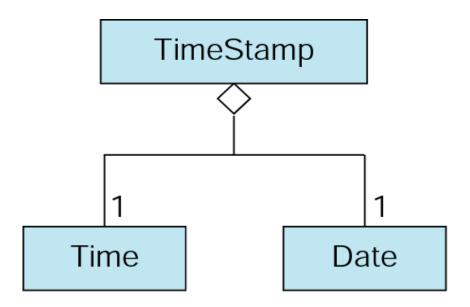






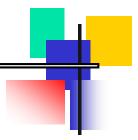
## Figure 10-21 Aggregation

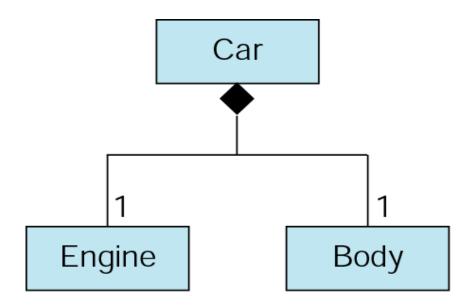






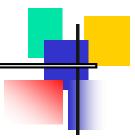
## Figure 10-22 Composition

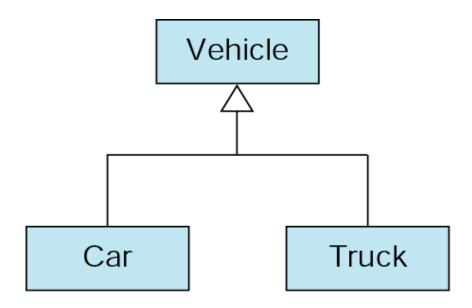






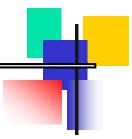
## Figure 10-23 Generalization diagram







## Figure 10-24 Object diagram



fr: Fraction

fr: Fraction

numerator denominator

