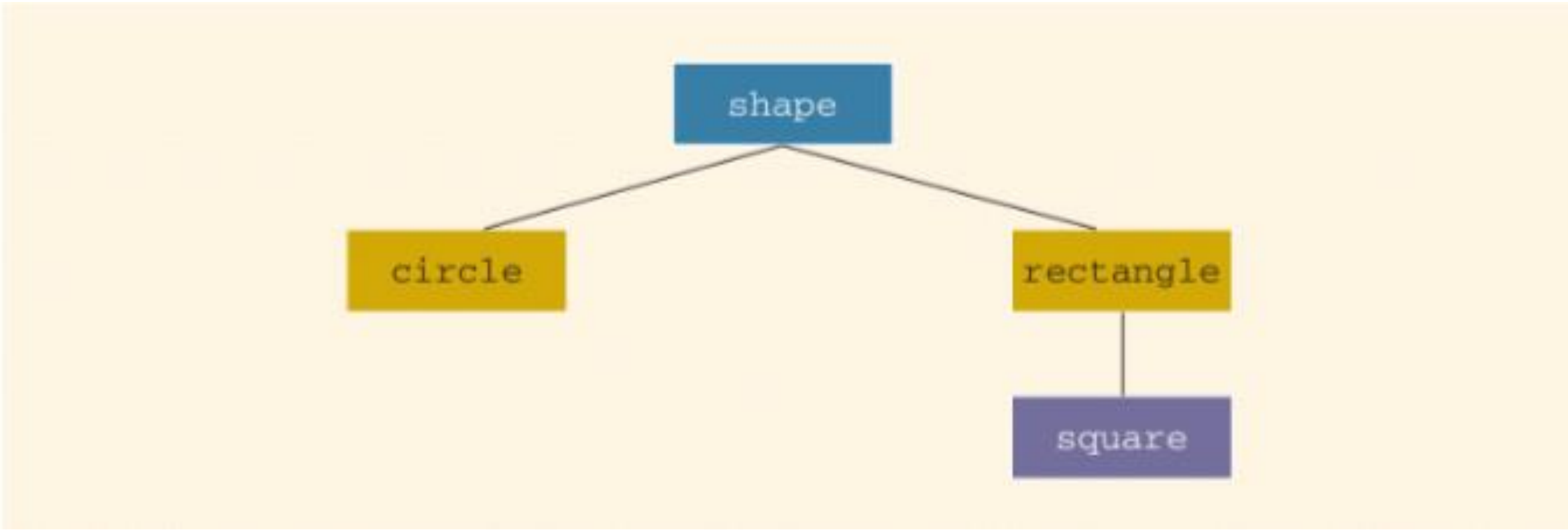
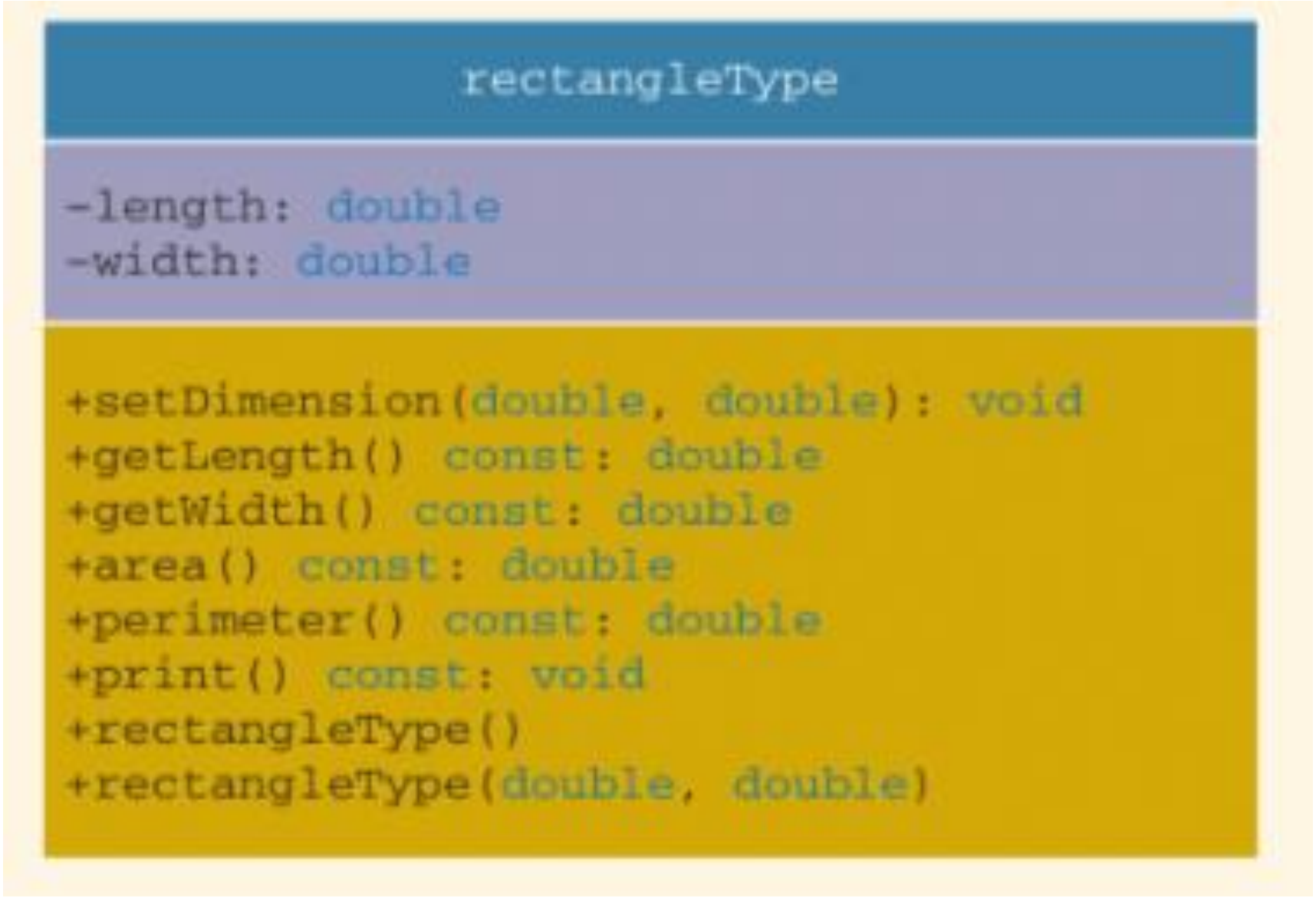

shape
circle
rectangle
square

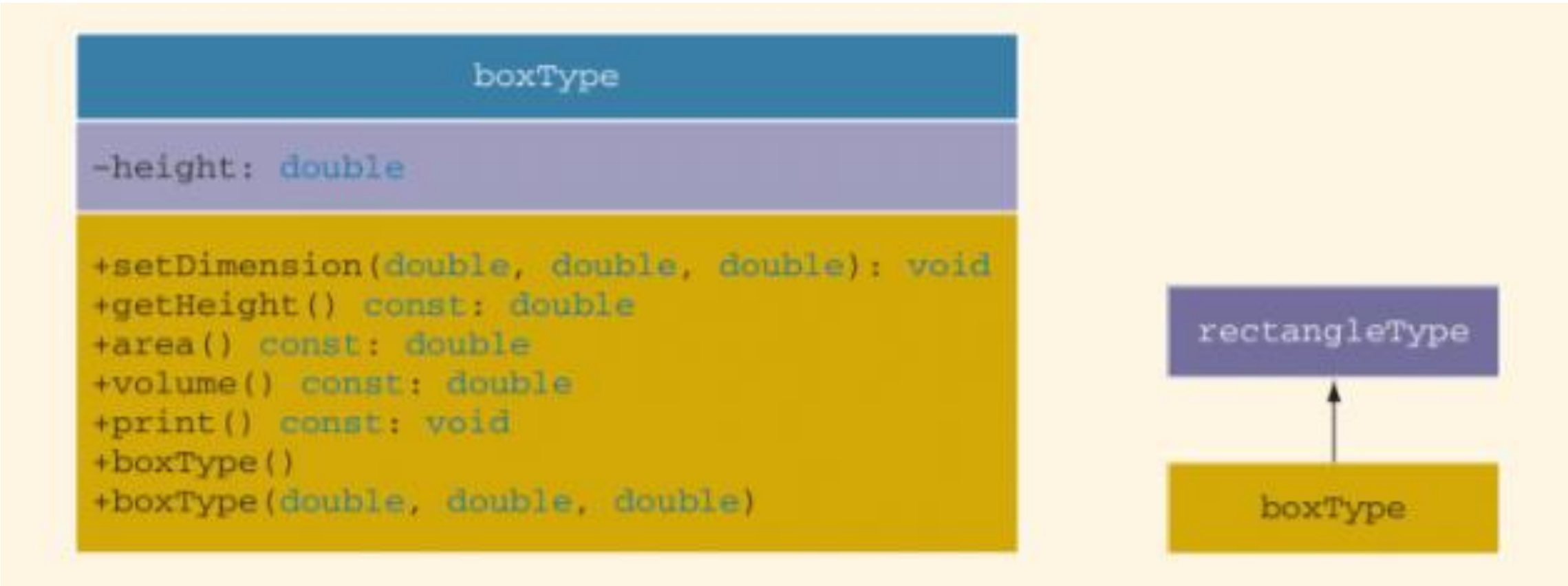
```
class className: memberAccessSpecifier baseClassName
{
    member list
};
```

```
class circle: public shape
{
      .
      .
      .
};
```

```
class circle: private shape
{
      .
      .
      .
};
```

```
class circle: shape
{
      .
      .
      .
};
```


rectangleType
-length: double
-width: double
+setDimension(double, double): void
+getLength() const: double
+getWidth() const: double
+area() const: double
+perimeter() const: double
+print() const: void
+rectangleType()
+rectangleType(double, double)


boxType
-height: double
+setDimension(double, double, double): void
+getHeight() const: double
+area() const: double
+volume() const: double
+print() const: void
+boxType()
+boxType(double, double, double)
rectangleType
boxType

```
void boxType::setDimension(double l, double w, double h)
{
    rectangleType::setDimension(l, w);

    if (h >= 0)
        height = h;
    else
        height = 0;
}
```

1. If `memberAccessSpecifier` is `public`—that is, the inheritance is `public`—then:
   a. The `public` members of `A` are `public` members of `B`. They can be directly accessed in `class B`.
   b. The `protected` members of `A` are `protected` members of `B`. They can be directly accessed by the member functions (and `friend` functions) of `B`.
   c. The `private` members of `A` are hidden in `B`. They cannot be directly accessed in `B`. They can be accessed by the member functions (and `friend` functions) of `B` through the `public` or `protected` members of `A`.

2. If `memberAccessSpecifier` is `protected`—that is, the inheritance is `protected`—then:

   a. The `public` members of `A` are `protected` members of `B`. They can be accessed by the member functions (and `friend` functions) of `B`.

   b. The `protected` members of `A` are `protected` members of `B`. They can be accessed by the member functions (and `friend` functions) of `B`.

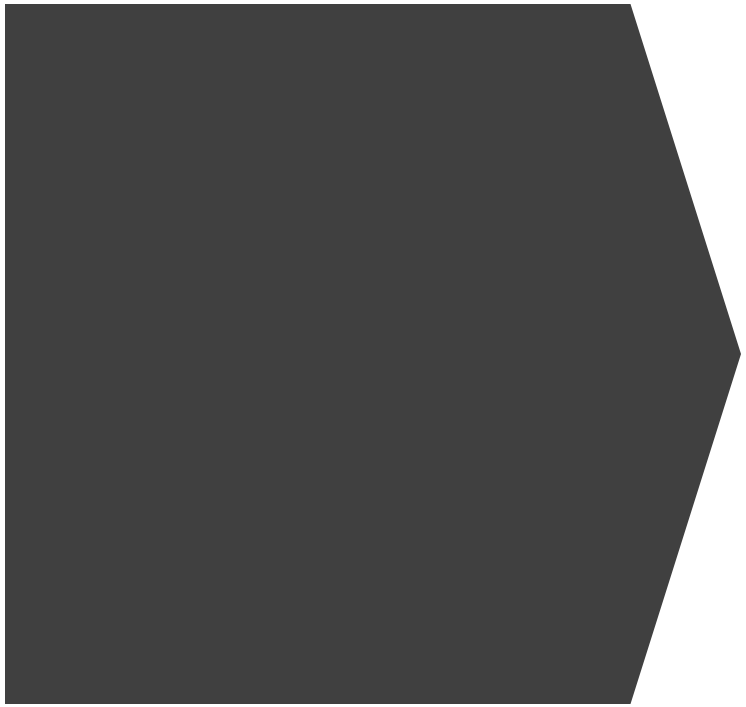
   c. The `private` members of `A` are hidden in `B`. They cannot be directly accessed in `B`. They can be accessed by the member functions (and `friend` functions) of `B` through the `public` or `protected` members of `A`.

3. If `memberAccessSpecifier` is `private`—that is, the inheritance is `private`—then:

   a. The `public` members of `A` are `private` members of `B`. They can be accessed by the member functions (and `friend` functions) of `B`.

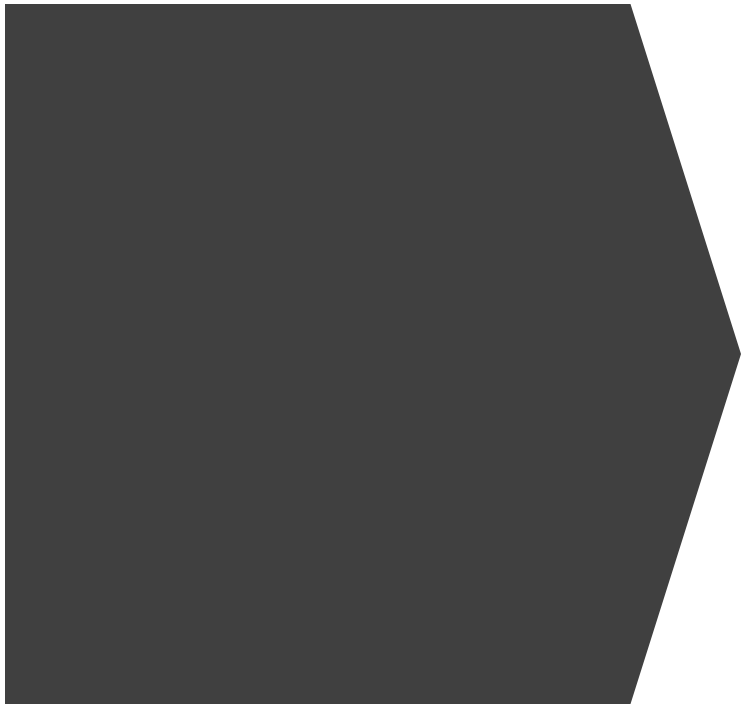
   b. The `protected` members of `A` are `private` members of `B`. They can be accessed by the member functions (and `friend` functions) of `B`.

   c. The `private` members of `A` are hidden in `B`. They cannot be directly accessed in `B`. They can be accessed by the member functions (and `friend` functions) of `B` through the `public` or `protected` members of `A`.



QUICK REVIEW

1. Inheritance and composition (aggregation) are meaningful ways to relate two or more classes.
2. Inheritance is an "is-a" relation.
3. Composition (aggregation) is a "has-a" relation.
4. In a single inheritance, the derived class is derived from only one existing class called the base class.
5. In a multiple inheritance, a derived class is derived from more than one base class.
6. The `private` members of a base class are `private` to the base class. The derived class cannot directly access them.
7. The `public` members of a base class can be inherited either as `public` or `private` by the derived class.
8. A derived class can redefine the member functions of a base class, but this redefinition applies only to the objects of the derived class.
9. A call to a base class's constructor (with parameters) is specified in the heading of the definition of the derived class's constructor.



10. If in the heading of the definition of a derived class's constructor, no call to a constructor (with parameters) of a base class is specified, then during the derived class's object declaration and initialization, the default constructor (if any) of the base class executes.
11. When initializing the object of a derived class, the constructor of the base class is executed first.
12. Review the inheritance rules given in this chapter.
13. In composition (aggregation), a member of a class is an object of another class.
14. In composition (aggregation), a call to the constructor of the member objects is specified in the heading of the definition of the class's constructor.
15. The three basic principles of OOD are encapsulation, inheritance, and polymorphism.
16. An easy way to identify classes, objects, and operations is to describe the problem in English and then identify all of the nouns and verbs. Choose your classes (objects) from the list of nouns and operations from the list of verbs.