BAKI ALİ NEFT MƏKTƏBİ
BAKU HIGHER OIL SCHOOL

# Programming and Computer Applications-2

# Object-Oriented Programming: Inheritance

Instructor : PhD, Associate Professor Leyla Muradkhanli

# Outline

- **Introduction**
- **Base Classes and Derived Classes**
- `public`, `protected` **and** `private` **Inheritance**
- **Multiple Inheritances**

# Introduction

- Inheritance is a form of software reuse in which you create a class that absorbs an existing class's data and behaviors and enhances them with new capabilities.
- The new class should inherit the members of an existing class.
- This existing class is called the base class, and the new class is referred to as the derived class.
- C++ offers `public`, `protected` and `private` inheritance.
- *With* `public` *inheritance, every object of a derived class is also an object of that derived class's base class.*
- However, base-class objects are not objects of their derived classes.

# Introduction (cont.)

- We distinguish between the *is-a* relationship and the *has-a* relationship.
- The *is-a* relationship represents inheritance.
- In an *is-a* relationship, an object of a derived class also can be treated as an object of its base class.
- By contrast, the *has-a* relationship represents composition.

# Base Classes and Derived Classes

- Figure 1 lists several simple examples of base classes and derived classes.
  - Base classes tend to be *more general* and derived classes tend to be *more specific*.
- Because every derived-class object *is an* object of its base class, and one base class can have *many* derived classes, the set of objects represented by a base class typically is *larger* than the set of objects represented by any of its derived classes.
- Inheritance relationships form class hierarchies.

| Base class | Derived classes |
|---|---|
| Student | GraduateStudent, UndergraduateStudent |
| Shape | Circle, Triangle, Rectangle, Sphere, Cube |
| Loan | CarLoan, HomeImprovementLoan, MortgageLoan |
| Employee | Faculty, Staff |
| Account | CheckingAccount, SavingsAccount |

**Fig. 1** Inheritance examples.

# Base Classes and Derived Classes (cont.)

- With single inheritance, a class is derived from *one* base class.
- With multiple inheritance, a derived class inherits simultaneously from *two or more* base classes.

# Base Classes and Derived Classes (cont.)

## *`Shape` Class Hierarchy*

- Consider the `Shape` inheritance hierarchy in Fig. 2.
- Begins with base class `Shape`.
- Classes `TwoDimensionalShape` and `ThreeDimensionalShape` derive from base class `Shape`—`Shape`s are either `TwoDimensionalShape`s or `Three-DimensionalShape`s.
- The third level of this hierarchy contains some more specific types of `TwoDimensionalShape`s and `ThreeDimensionalShape`s.

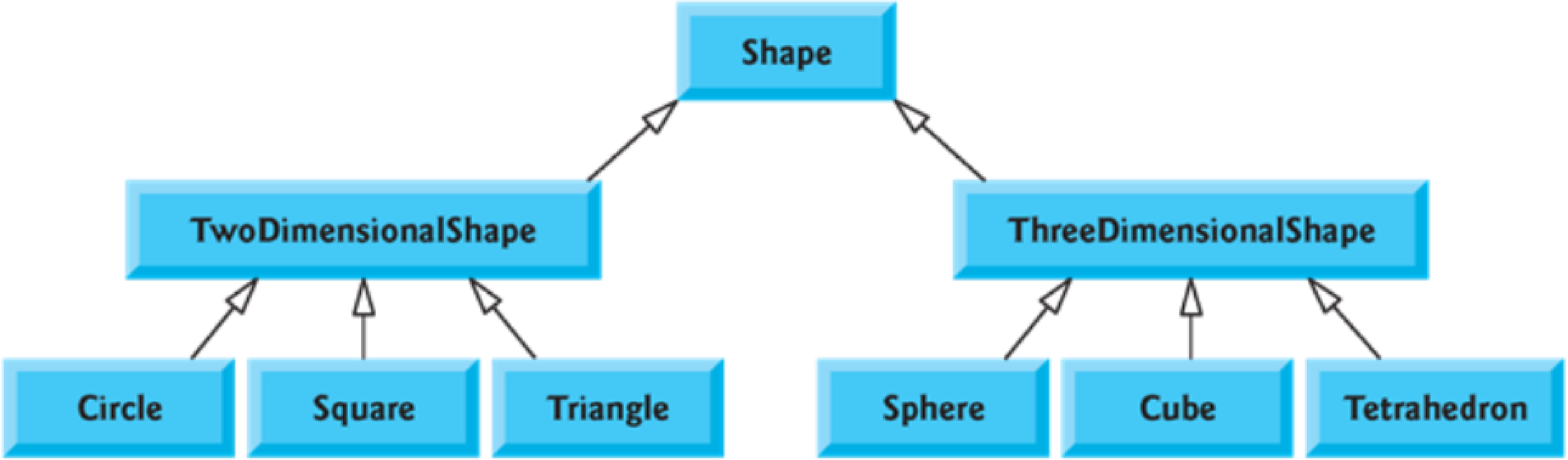


Fig. 2 Inheritance hierarchy for Shapes.

# Base Classes and Derived Classes (cont.)

A class can be derived from more than one classes, which means it can inherit data and functions from multiple base classes. To define a derived class, we use a class derivation list to specify the base class(es). A class derivation list names one or more base classes and has the form:

```
class derived-class: access-specifier base-class
```

Where access-specifier is one of **public, protected,** or **private**, and base-class is the name of a previously defined class. If the access-specifier is not used, then it is private by default.

Consider a base class **Shape** and its derived class **Rectangle** as follows:

# Base Classes and Derived Classes

```
#include <iostream>
using namespace std;
// Base class
class Shape
{
   public:
      void setWidth(int w)
      {
         width = w;
      }
      void setLength(int l)
      {
         length = l;
      }
   protected:
      int width;
      int length;
};
```

# Base Classes and Derived Classes

```
// Derived class
class Rectangle: public Shape
{
   public:
      int Area()
      {
         return (width * length);
      }
}; int main()
{
   Rectangle Rec;
   Rec.setWidth(5);
   Rec.setLength(7);
   // Print the area of the object.
   cout << "Total area: " << Rec.Area() << endl;
   return 0;
}
```

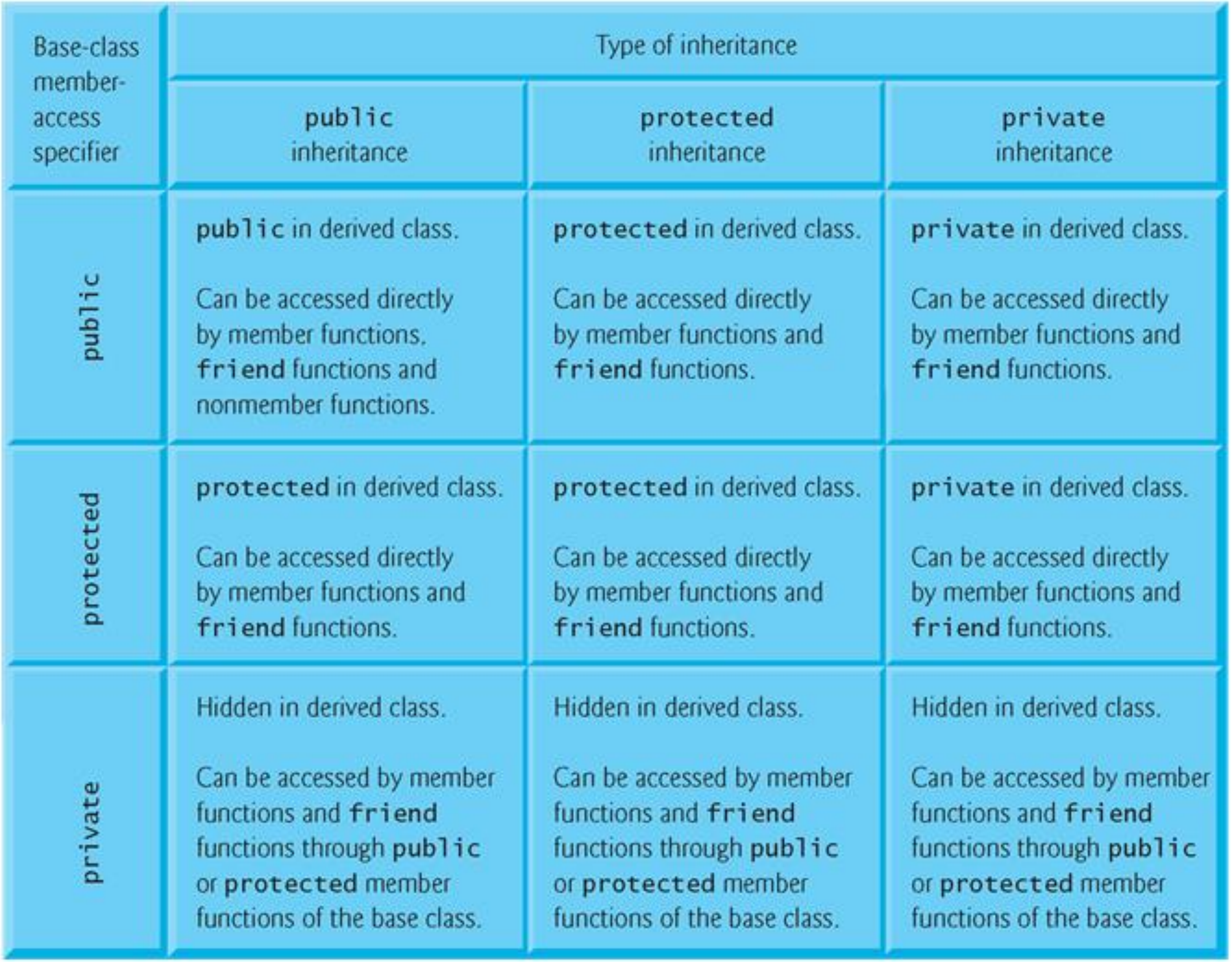
# Base Classes and Derived Classes

When the above code is compiled and executed, it produces the following result:

```
Total area: 35
```

# `public`, `protected` and `private` Inheritance

- When deriving a class from a base class, the base class may be inherited through `public`, `protected` or `private` inheritance.
- Use of `protected` and `private` inheritance is rare.
- Figure  summarizes for each type of inheritance the accessibility of base-class members in a derived class.
- The first column contains the base-class access specifiers.
- A base class's `private` members are *never* accessible directly from a derived class, but can be accessed through calls to the `public` and `protected` members of the base class.



| Base-class member-access specifier | Type of inheritance | | |
|---|---|---|---|
| | `public` inheritance | `protected` inheritance | `private` inheritance |
| `public` | `public` in derived class.<br>Can be accessed directly by member functions, `friend` functions and nonmember functions. | `protected` in derived class.<br>Can be accessed directly by member functions and `friend` functions. | `private` in derived class.<br>Can be accessed directly by member functions and `friend` functions. |
| `protected` | `protected` in derived class.<br>Can be accessed directly by member functions and `friend` functions. | `protected` in derived class.<br>Can be accessed directly by member functions and `friend` functions. | `private` in derived class.<br>Can be accessed directly by member functions and `friend` functions. |
| `private` | Hidden in derived class.<br>Can be accessed by member functions and `friend` functions through `public` or `protected` member functions of the base class. | Hidden in derived class.<br>Can be accessed by member functions and `friend` functions through `public` or `protected` member functions of the base class. | Hidden in derived class.<br>Can be accessed by member functions and `friend` functions through `public` or `protected` member functions of the base class. |

# Multiple Inheritances

A C++ class can inherit members from more than one class and here is the extended syntax:

```
class derived-class: access baseA, access baseB....
```

Where access is one of **public, protected,** or **private** and would be given for every base class and they will be separated by comma as shown above. Let us try the following example:

# Multiple Inheritances

```
#include <iostream>
using namespace std;
// Base class Shape
class Shape
{
   public:
      void setWidth(int w)
      {
         width = w;
      }
      void setLength(int l)
      {
         length = l;
      }
   protected:
      int width;
      int length;
};
```

# Multiple Inheritances

```
// Base class PaintCost
class PaintCost
{
   public:
      int Cost(int area)
      {
         return area * 50;
      }
};
// Derived class
class Rectangle: public Shape, public PaintCost
{
   public:
      int Area()
      {
         return (width * length);
      }
};
```

# Multiple Inheritances

```
int main()
{
    Rectangle Rec;
    int area;

    Rec.setWidth(10);
    Rec.setLength(7);

    area = Rec.Area();

    // Print the area of the object.
    cout << "Total area: " << Rec.Area() << endl;

    // Print the total cost of painting
    cout << "Total paint cost: $" << Rec.Cost(area) << endl;

    return 0;
}
```

# Multiple Inheritances

When the above code is compiled and executed, it produces the following result:

```
Total area: 70
Total paint cost: $3500
```