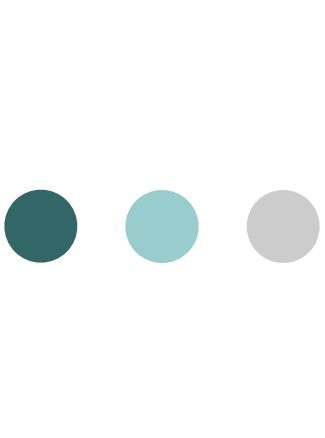


15.2

Protected Members and Class Access

Protected Members and Class Access

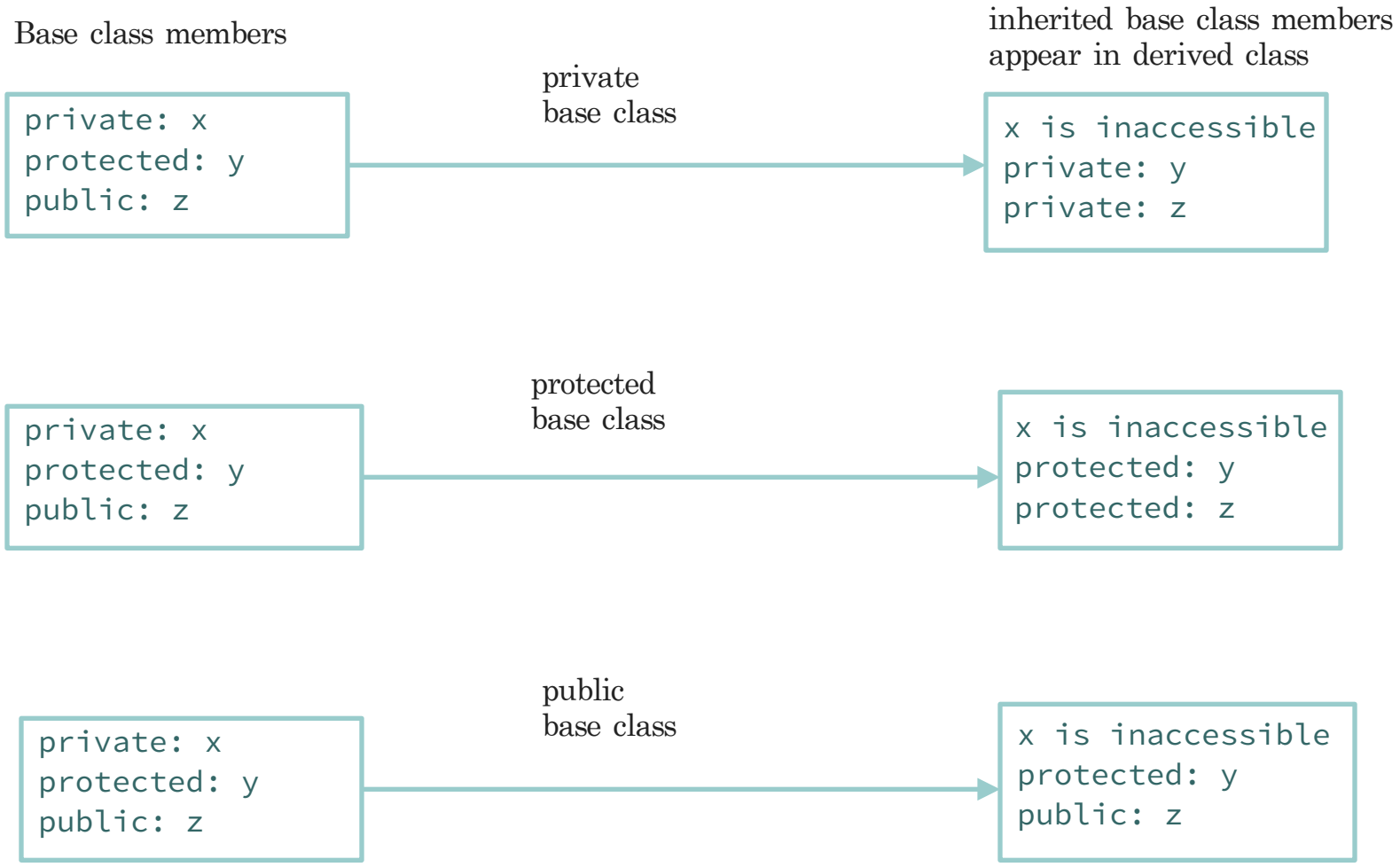
- protected member access specification: like private, but accessible by the member functions of the derived class

- Class access specification: determines how private, protected, and public members of base class are inherited by the derived class

Class Access Specifiers

- public parent class resources – derived class member functions and the object of the derived class can access all of them directly (not vice-versa)
- protected parent class resources – more restrictive than public, derived class member functions can access them directly. However, not accessible to the object of the derived class.
- private parent class resources – not accessible by the member functions of the derived class or the object of the derived class.

Inheritance vs. Access



More Inheritance vs. Access

```
class Grade
private members:
  char letter;
  float score;
  void calcGrade();

public members:
  void setScore(float);
  float getScore();
  char getLetter();
```

```
class Test: public Grade
private members:
  int numQuestions;
  float pointsEach;
  int numMissed;

public members:
  Test(int, int);
```

When Test class inherits from Grade class using public class access, it looks like this:


```
private members:
  int numQuestions:
  float pointsEach;
  int numMissed;

public members:
  Test(int, int);
  void setScore(float);
  float getScore();
  float getLetter();
```

More Inheritance vs. Access (2)

```
class Grade
private members:
  char letter;
  float score;
  void calcGrade();

public members:
  void setScore(float);
  float getScore();
  char getLetter();
```

```
class Test: protected Grade
private members:
  int numQuestions;
  float pointsEach;
  int numMissed;

public members:
  Test(int, int);
```

When Test class inherits from Grade class using protected class access, it looks like this:

```
private members:
  int numQuestions:
  float pointsEach;
  int numMissed;

public members:
  Test(int, int);
protected members:
  void setScore(float);
  float getScore();
  float getLetter();
```

More Inheritance vs. Access (3)

```
class Grade

private members:
  char letter;
  float score;
  void calcGrade();
public members:
  void setScore(float);
  float getScore();
  char getLetter();
```

```
class Test: private Grade

private members:
  int numQuestions;
  float pointsEach;
  int numMissed;
public members:
  Test(int, int);
```

When Test class inherits from Grade class using private class access, it looks like this:

```
private members:
  int numQuestions:
  float pointsEach;
  int numMissed;
  void setScore(float);
  float getScore();
  float getLetter();
public members:
  Test(int, int);
```

15.3

Constructors and Destructors in Base and Derived Classes

Constructors and Destructors in Base and Derived Classes

- Derived classes can have their own constructors and destructors
- When an object of a derived class is created,
  - the base class's constructor is executed first
  - followed by the derived class's constructor
- When an object of a derived class is destroyed,
  - derived class's destructor is called first
  - followed by the base class's destructor

Constructors and Destructors in Base and Derived Classes

Program 15-4

```
1  // This program demonstrates the order in which base and
2  // derived class constructors and destructors are called.
3  #include <iostream>
4  using namespace std;
5
6  //*********************************
7  // BaseClass declaration          *
8  //*********************************
9
```

Constructors and Destructors in Base and Derived Classes

Program 15-4 *(continued)*

```
10  class BaseClass
11  {
12  public:
13     BaseClass()  // Constructor
14        { cout << "This is the BaseClass constructor.\n"; }
15
16     ~BaseClass() // Destructor
17        { cout << "This is the BaseClass destructor.\n"; }
18  };
19
20  //*********************************
21  // DerivedClass declaration      *
22  //*********************************
23
24  class DerivedClass : public BaseClass
25  {
26  public:
27     DerivedClass()  // Constructor
28        { cout << "This is the DerivedClass constructor.\n"; }
29
30     ~DerivedClass()  // Destructor
31        { cout << "This is the DerivedClass destructor.\n"; }
32  };
33
```

Constructors and Destructors in Base and Derived Classes

```
34  //*********************************
35  // main function                  *
36  //*********************************
37
38  int main()
39  {
40     cout << "We will now define a DerivedClass object.\n";
41
42     DerivedClass object;
43
44     cout << "The program is now going to end.\n";
45     return 0;
46  }
```

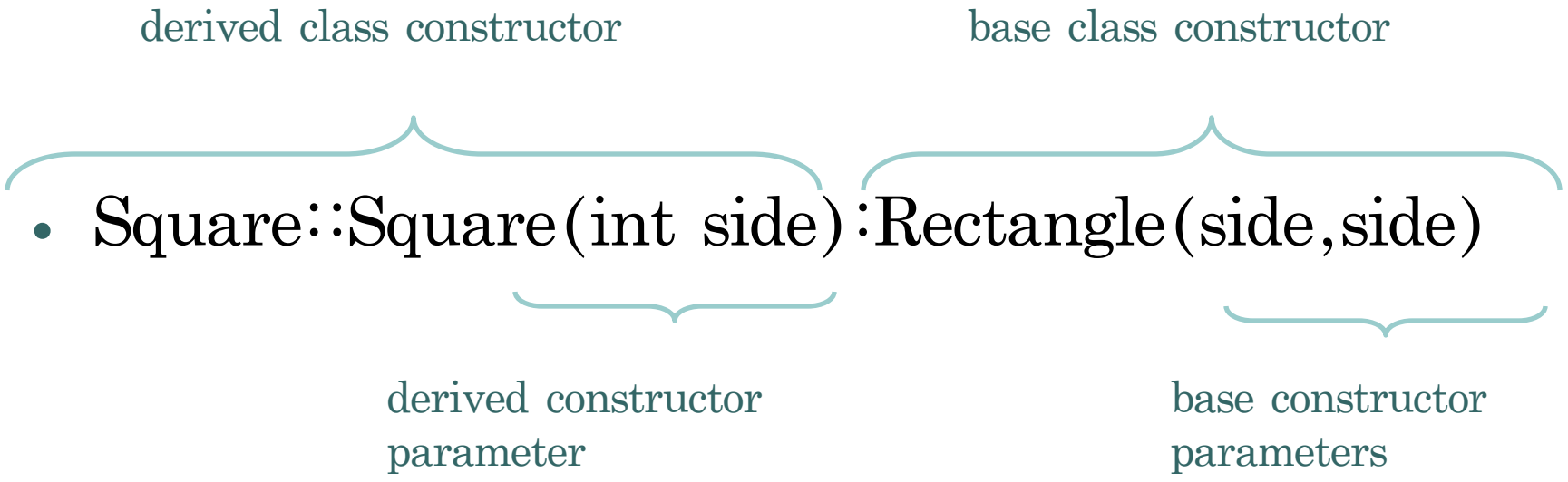
Program Output

```
We will now define a DerivedClass object.
This is the BaseClass constructor.
This is the DerivedClass constructor.
The program is now going to end.
This is the DerivedClass destructor.
This is the BaseClass destructor.
```

Passing Arguments to Base Class Constructor

- Allows selection between multiple base class constructors
- Specify arguments to base constructor on derived constructor heading:
  - `Square::Square(int side) : Rectangle(side, side)`
- Can also be done with inline constructors
- Must be done if base class has no default constructor

Passing Arguments to Base Class Constructor



Constructor Inheritance

- In a derived class, some constructors can be inherited from the base class.
- The constructors that cannot be inherited are:
  - the default constructor
  - the copy constructor

Constructor Inheritance

- Consider the following:

```
class MyBase
{
private:
    int ival;
    double dval;
public:
    MyBase(int i)
    { ival = i; }

    MyBase(double d)
    { dval = d; }
};
```

```
class MyDerived : MyBase
{
public:
    MyDerived(int i) : MyBase(i)
    {}

    MyDerived(double d) : MyBase(d)
    {}
};
```

Constructor Inheritance

- We can rewrite the MyDerived class:

```
class MyBase
{
private:
    int ival;
    double dval;
public:
    MyBase(int i)
    { ival = i; }

    MyBase(double d)
    { dval = d; }
};
```

```
class MyDerived : MyBase
{
    using MyBase::MyBase;
};
```

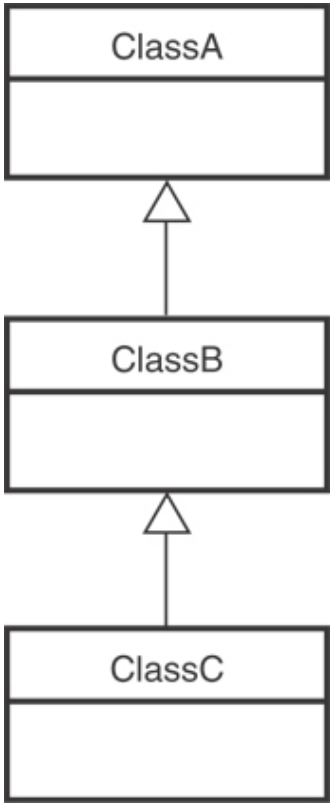
The using statement causes the class to inherit the base class constructors.

15.5

Class Hierarchies

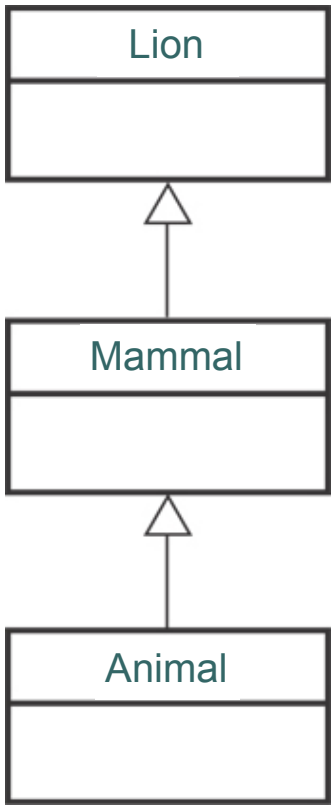
Class Hierarchies

- A base class can be derived from another base class.



Class Hierarchies (cont)

- Consider the Animal, Mammal, and the Lion class hierarchy.



Class Hierarchies (cont)

```
5   class Animal
6   {    string animalSound;
7   public:
8       Animal()
9       {    animalSound = "Awwww";
10          cout<<"\nI am an animal and I sound: "
11              <<getAnimalSound();
12      }
13      string getAnimalSound() const { return animalSound; }
14  };
```

Class Hierarchies (cont)

```
17  class Mammal : public Animal
18  {
19      string mammalSound;
20
21  public:
22      Mammal() : Animal()
23      {    mammalSound = "Meeeoowww";
24          cout<< "\nI am a Mammal and I sound: "
25              <<getMammalSound();
26      }
27      string getMammalSound() const { return mammalSound; }
28  };
```

Class Hierarchies (cont)

```
31  class Lion : public Mammal
32  {   string lionSound;
33  public:
34      Lion() : Mammal()
35      {   lionSound = "Raawwrrrrr";
36          cout<< "\nI am a Lion and I sound: "
37              <<getLionSound();
38      }
39      string getLionSound() const { return lionSound; }
40  };
41
42  int main() // testing the derived class
43  {   Lion theKing;
44      cout << "\nAnimal Sound is: " << theKing.getAnimalSound();
45      cout << "\nMammal Sound is: " << theKing.getMammalSound();
46      cout << "\nLion Sound is: " << theKing.getLionSound();
47  }
```