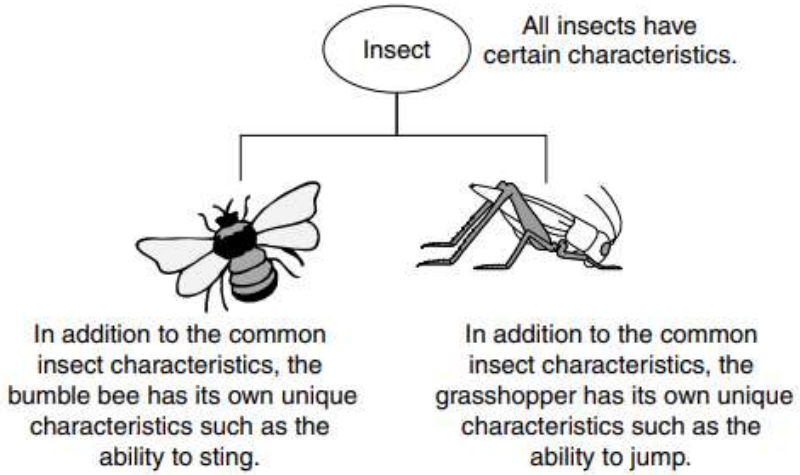
Lecture 12
OOP

Khola Naseem
khola.naseem@uet.edu.pk

Inheritance

- Inheritance?
- Inheritance allows a new class to be based on an existing class. The new class inherits all the member variables and functions (except the constructors and destructor) of the class it is based on.
- Generalization and Specialization

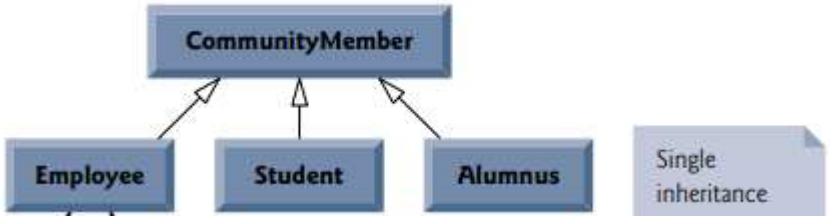


Credit: Khola Naseem

Inheritance

- Inheritance Examples:

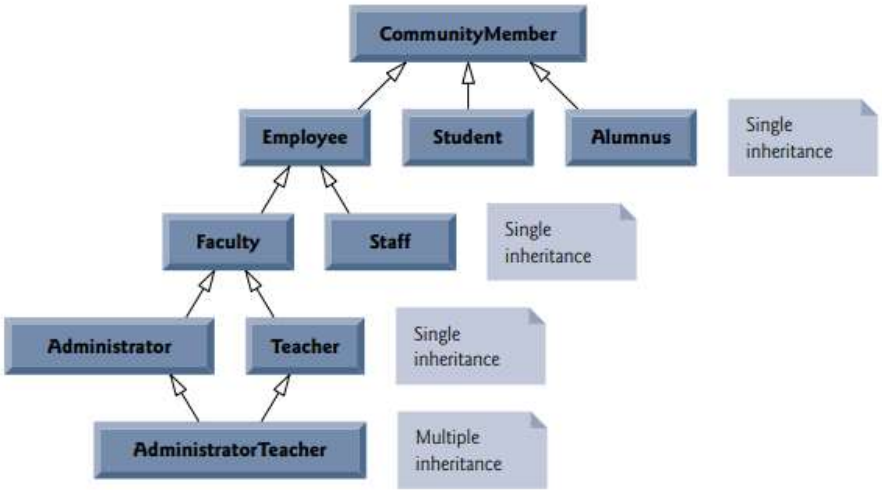
Base class	Derived classes
Student	GraduateStudent, UndergraduateStudent
Shape	Circle, Triangle, Rectangle, Sphere, Cube
Loan	CarLoan, HomeImprovementLoan, MortgageLoan
Employee	Faculty, Staff
Account	CheckingAccount, SavingsAccount



Credit: Khola Naseem

Inheritance

- Inheritance

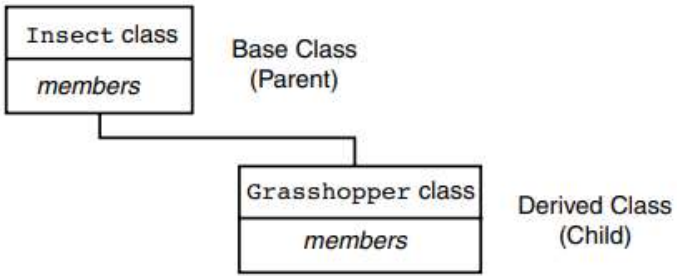


- With single inheritance, a class is derived from one base class.
- With multiple inheritance, a derived class inherits simultaneously from two or more
- direct base class
- Indirect base class

Credit: Khola Naseem

Inheritance

- Inheritance and the "**Is a**" Relationship
- When one object is a specialized version of another object, there is an "is a" relationship between them. For example, a grasshopper is an insect
- Examples:
  - A poodle is a dog.
  - A car is a vehicle.
  - A tree is a plant.
  - A rectangle is a shape.
  - A football player is an athlete
- Inheritance involves a base class and a derived class. The base class is the general class and the derived class is the specialized class.



Credit: Khola Naseem

Inheritance

- Inheritance:
  - 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.
- Example:

```
class Shape
{

};
class TwoDimensionalShape : public Shape
{

}
```

Inheritance

- Example:

```
class Shape
{
    public:
    void displayshape() {
        cout << "i am function of shape class" << endl;
    }

};
class TwoDimensionalShape : public Shape
{
    public:
     void displayTshape() {
        cout << "i am function of TwoDimensionalShape class" << endl;
    }

};


int main(int argc, char** argv) {
    TwoDimensionalShape t;
    t.displayshape();
    t.displayTshape();
}
```

- Output:

```
i am function of shape class
i am function of TwoDimensionalShape class
```

Credit: Khola Naseem

The access modifier protected is especially relevant when it comes to C++ inheritance.

Inheritance

- **Access Modes of Inheritance in C++:**
- This is an example of public inheritance, the most commonly used form. The other types are private inheritance and protected inheritance
- With all forms of inheritance, private members of a base class are not accessible directly from that class's derived classes, but these private base-class members are still inherited
- With **public inheritance**, all other base-class members retain their original member access when they become members of the derived class
  - e.g., public members of the base class become public members of the derived class, and, protected members of the base class become protected members of the derived class
- With **private inheritance ,**all the members of the base class become private members in the derived class.

```
class TwoDimensionalShape : private Shape
```

- With **protected inheritance** The public members of the base class become protected members in the derived class.

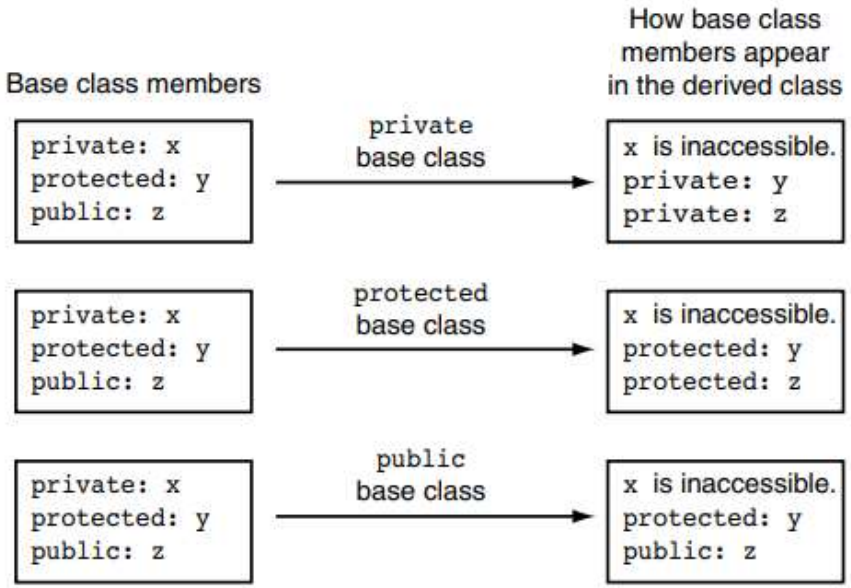
```
class TwoDimensionalShape : protected Shape
```

Credit: Khola Naseem

The access modifier protected is especially relevant when it comes to C++ inheritance.

Inheritance

- **Access Modes of Inheritance in C++:**
- With **public inheritance**, all other base-class members retain their original member access when they become members of the derived class
- With **private inheritance ,**all the members of the base class become private members in the derived class.
- With **protected inheritance** The public members of the base class become protected members in the derived class.



Credit: Khola Naseem

The access modifier protected is especially relevant when it comes to C++ inheritance.

Inheritance

- **Protected access modifier :**
- The access modifier protected is especially relevant to inheritance.
- Like private members, protected members are inaccessible outside of the class. However, they can be accessed by derived classes and friend classes/functions.
- We need protected members if we want to hide the data of a class, but still want that data to be inherited by its derived classes.

```
class Shape
{
    private:
       string color;
    protected:
      int Dim;
    public:
    void displayshape() {
        cout << "i am function of shape class" << endl;
    }
    void setcolor(string col)
    {
        color=col;
    }
    string getcolor()
    {
        return color;
    }

};
```

Credit: Khola Naseem

The access modifier protected is especially relevant when it comes to C++ inheritance.

Inheritance

- **Protected access modifier :**

```
class Shape
{
    private:
       string color;
    protected:
      int Dim;
    public:
    void displayshape() {
        cout << "i am function of shape class" << endl;
    }
    void setcolor(string col)
    {
        color=col;
    }
    string getcolor()
    {
        return color;
    }

};
class TwoDimensionalShape : public Shape

{
    public:
     void displayTshape() {
        cout << "i am function of TwoDimensionalShape class" << endl;
    }
    void setDim(int a)
    {
        Dim=a;
    }
    void displayData(string c) {
        cout << "Color of the shape is : " <<c <<" and Dimension is :  " <<Dim<< endl;
    }

};
```

The access modifier protected is especially relevant when it comes to C++ inheritance.

Inheritance

- **Protected access modifier :**

```
class Shape
{
    private:
       string color;
    protected:
      int Dim;
    public:
    void displayshape() {
        cout << "i am function of shape class" << endl;
    }
    void setcolor(string col)
    {
        color=col;
    }
    string getcolor()
    {
        return color;
    }

};
class TwoDimensionalShape : public Shape

{
    public:
     void displayTshape() {
        cout << "i am function of TwoDimensionalShape class" << endl;
    }
    void setDim(int a)
    {
        Dim=a;
    }
    void displayData(string c) {
        cout << "Color of the shape is : " <<c <<" and Dimension is :  " <<Dim<< endl;
    }

};
```

```
int main(int argc, char** argv) {
    TwoDimensionalShape t;
    t.setcolor("Red");
    t.displayshape();
    t.displayTshape();
    t.setDim(2);
    t.displayData(t.getcolor());

}
```

Credit: Khola Naseem

The access modifier protected is especially relevant when it comes to C++ inheritance.

Inheritance

- **Protected access modifier :**

```
int main(int argc, char** argv) {
    TwoDimensionalShape t;
    t.setcolor("Red");
    t.displayshape();
    t.displayTshape();
    t.setDim(2);
    t.displayData(t.getcolor());

}
```

- **Output:**

```
i am function of shape class
i am function of TwoDimensionalShape class
Color of the shape is : Red and Dimension is :  2
```

The access modifier protected is especially relevant when it comes to C++ inheritance.

Inheritance

- **Access Modes of Inheritance in C++:**
- With **private inheritance ,**all the members of the base class become private members in the derived class.

```
class Shape
{
    private:
       string color;
    protected:
      int Dim;
    public:
    void displayshape() {
        cout << "i am function of shape class" << endl;
    }
    void setcolor(string col)
    {
        color=col;
    }
    string getcolor()
    {
        return color;
    }
};
class TwoDimensionalShape : private Shape

{
    public:
     void displayTshape() {
        cout << "i am function of TwoDimensionalShape class" << endl;
        displayshape();
    }
    void setDim(int a)
    {
        Dim=a;
    }
    void displayData(string c) {
        cout << "Color of the shape is : " <<c <<" and Dimension is :  " <<Dim<< endl;
    }

};
int main(int argc, char** argv) {
    TwoDimensionalShape t;
//  t.setcolor("Red");
//t.displayshape();
    t.displayTshape();
    t.setDim(2);
    t.displayData("red");
}
```

Output:

```
i am function of TwoDimensionalShape class
i am function of shape class
Color of the shape is : red and Dimension is :  2
```

Credit: Khola Naseem

The access modifier protected is especially relevant when it comes to C++ inheritance.

Inheritance

- **Access Modes of Inheritance in C++:**
- With **protected inheritance**

```
class Shape
{
    private:
       string color;
    protected:
      int Dim;
    public:
    void displayshape() {
        cout << "i am function of shape class" << endl;
    }
    void setcolor(string col)
    {
        color=col;
    }
    string getcolor()
    {
        return color;
    }
};
class TwoDimensionalShape : protected Shape
{
    public:
     void displayTshape() {
        cout << "i am function of TwoDimensionalShape class" << endl;
        displayshape();
    }
    void setDim(int a)
    {
        Dim=a;
    }
    void displayData(string c) {
        cout << "Color of the shape is : " <<c <<" and Dimension is :  " <<Dim<< endl;
    }

};
int main(int argc, char** argv) {
    TwoDimensionalShape t;
//  t.setcolor("Red");
//t.displayshape();
    t.displayTshape();
    t.setDim(2);
    t.displayData("red");
}
```

Output:

```
i am function of TwoDimensionalShape class
i am function of shape class
Color of the shape is : red and Dimension is :  2
```

Credit: Khola Naseem

The access modifier protected is especially relevant when it comes to C++ inheritance.

Inheritance

- **Access Modes of Inheritance in C++:**
- With **protected inheritance**

```
class Shape
{
    private:
       string color;
    protected:
      int Dim;
    public:
    void displayshape() {
        cout << "i am function of shape class" << endl;
    }
    void setcolor(string col)
    {
        color=col;
    }
    string getcolor()
    {
        return color;
    }
};
class TwoDimensionalShape : protected Shape
{
    public:
     void displayTshape() {
        cout << "i am function of TwoDimensionalShape class" << endl;
        displayshape();
    }
    void setDim(int a)
    {
        Dim=a;
    }
    void displayData(string c) {
        cout << "Color of the shape is : " <<c <<" and Dimension is :  " <<Dim<< endl;
    }
};
class box: public TwoDimensionalShape
{
    public:
    void bDisplay()
    {
    setcolor("red");
    setDim(2);
    displayData(getcolor());
    }
};
```

```
int main(int argc, char** argv) {
    box b;
    b.bDisplay();
    //TwoDimensionalShape t;
    //t.setcolor("Red");
    //t.displayshape();
    //t.displayTshape();
    //t.setDim(2);
    //t.displayData("red");
}
```

Output:

```
Color of the shape is : red and Dimension is :  2
```

Credit: Khola Naseem

The access modifier protected is especially relevant when it comes to C++ inheritance.

Inheritance

- **Access Modes of Inheritance in C++:**
- With **private inheritance**

```
class Shape
{
    private:
       string color;
    protected:
      int Dim;
    public:
    void displayshape() {
        cout << "i am function of shape class" << endl;
    }
    void setcolor(string col)
    {
        color=col;
    }
    string getcolor()
    {
        return color;
    }
};
class TwoDimensionalShape : private Shape
{
    public:
     void displayTshape() {
        cout << "i am function of TwoDimensionalShape class" << endl;
        displayshape();
    }
    void setDim(int a)
    {
        Dim=a;
    }
    void displayData(string c) {
        cout << "Color of the shape is : " <<c <<" and Dimension is :  " <<Dim<< endl;
    }
};
```

```
class box: public TwoDimensionalShape
{
    public:
    void bDisplay()
    {
    setcolor("red");
    setDim(2);
    displayData(getcolor());
    }
};
int main(int argc, char** argv) {
    box b;
    b.bDisplay();
}
```

Credit: Khola Naseem

The access modifier protected is especially relevant when it comes to C++ inheritance.

Inheritance

- **Access Modes of Inheritance in C++:**
- With **private inheritance**

```
[Error] 'void Shape::setcolor(std::string)' is inaccessible
[Error] within this context
[Error] 'void Shape::setcolor(std::string)' is inaccessible
[Error] within this context
[Error] 'Shape' is not an accessible base of 'box'
[Error] 'std::string Shape::getcolor()' is inaccessible
[Error] within this context
[Error] 'std::string Shape::getcolor()' is inaccessible
[Error] within this context
```

```
class Shape
{
    private:
       string color;
    protected:
      int Dim;
    public:
    void displayshape() {
        cout << "i am function of shape class" << endl;
    }
    void setcolor(string col)
    {
        color=col;
    }
    string getcolor()
    {
        return color;
    }
};
class TwoDimensionalShape : private Shape
{
    public:
     void displayTshape() {
        cout << "i am function of TwoDimensionalShape class" << endl;
        displayshape();
    }
    void setDim(int a)
    {
        Dim=a;
    }
    void displayData(string c) {
        cout << "Color of the shape is : " <<c <<" and Dimension is :  " <<Dim<< endl;
    }
};
```

```
class box: public TwoDimensionalShape
{
    public:
    void bDisplay()
    {
    setcolor("red");
    setDim(2);
    displayData(getcolor());
    }
};
int main(int argc, char** argv) {
    box b;
    b.bDisplay();
}
```

Credit: Khola Naseem

The access modifier protected is especially relevant when it comes to C++ inheritance.

Inheritance

- **Constructor and destructor:**
- The base class's constructor is called before the derived class's constructor. The destructors are called in reverse order, with the derived class's destructor being called first.

```
#include <iostream>
using namespace std;

class BaseClass
 {
public:
    BaseClass() // Constructor
       { cout << "This is the BaseClass constructor.\n"; }
    ~BaseClass() // Destructor
    { cout << "This is the BaseClass destructor.\n"; }
  };
class DerivedClass : public BaseClass
  {
  public:
  DerivedClass()
       { cout << "This is the DerivedClass constructor.\n"; }
  ~DerivedClass() // Destructor
       { cout << "This is the DerivedClass destructor.\n"; }
  };
int main()
{
    cout << "We will now define a DerivedClass object.\n";
    DerivedClass object;
    cout << "The program is now going to end.\n";
    return 0;
}
```

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.
```

Credit: Khola Naseem

The access modifier protected is especially relevant when it comes to C++ inheritance.

Function overloading:

- If any class have multiple functions with same names but different parameters then they are said to be overloaded.
- Function overloading allows you to use the same name for different functions, to perform, either same or different functions in the same class

:

```
class Base {
   public:
      void printFunction(int i) {
        cout << "int value : " << i << endl;
      }
      void printFunction(double  d) {
        cout << "float value =  " << d << endl;
      }
      void printFunction(char* c) {
        cout << "characters: " << c << endl;
      }
};

int main() {
   Base b;
   b.printFunction(6);

   b.printFunction(340.263);

   b.printFunction("Hello world");

   return 0;
}
```

```
int value : 6
float value =  340.263
characters: Hello world
```

Credit: Khola Naseem

The access modifier protected is especially relevant when it comes to C++ inheritance.

Function overloading:

```cpp
#include <iostream>

using namespace std;

class parent_class

{
public:
    void display_message()
    {

        cout << "I am the base class function.\n";
    }
};

class derived_class : public parent_class

{

public:
    void display_message()

    {

        cout << "I am the derived class function.\n";

    }

};
int main()

{

    derived_class obj;
    obj.display_message();

    return 0;

}
```

```
I am the derived class function.
```

The access modifier protected is especially relevant when it comes to C++ inheritance.

Function overloading:

```cpp
#include <iostream>

using namespace std;

class parent_class

{
public:
    void display_message()
    {

        cout << "I am the base class function.\n";
    }
};

class derived_class : public parent_class

{

public:
    void display_message()

    {

        cout << "I am the derived class function.\n";

    }

};
int main()

{

    derived_class obj;
    obj.display_message();
    derived_class obj2;
     obj2.parent_class::display_message();

    return 0;
```

```
I am the derived class function.
I am the base class function.
```

Credit: Khola Naseem

The access modifier protected is especially relevant when it comes to C++ inheritance.

Function overloading:

- In inheritance

```
class Base {
   public:
      void printFunction(int i) {
        cout << "int value : " << i << endl;
      }
 };
class Drived:public Base {
   public:
      void printFunction(double  d) {
        cout << "float value =  " << d << endl;
      }
};
int main() {
   Drived d;
   d.printFunction(6);
   d.printFunction(340.263);
```

Output:

```
float value =  6
float value =  340.263
```

Credit: Khola Naseem

The access modifier protected is especially relevant when it comes to C++ inheritance.

Function overloading:

- In inheritance

```
class Base {
   public:
      void printFunction(int i) {
        cout << "int value : " << i << endl;
      }
 };
class Drived:public Base {
   public:
      using Base::printFunction;
      void printFunction(double  d) {
        cout << "float value =  " << d << endl;
      }
};
int main() {
   Drived d;
   d.printFunction(6);
   d.printFunction(340.263);
```

Output:

```
int value : 6
float value =  340.263
```

Reference material

- **For Practice Questions, refer to these books**
  - C++ Programming From Problem Analysis To Program Design, 5th Edition, D.S.Malik. Chapter 12.
  - C++ How to Program, Deitel & Deitel, 5th Edition, Prentice Hall.
  - Object Oriented Programming in C++ by Robert Lafore.
  - Object Oriented Software Construction, Bertrand Meyer's
  - Object-Oriented Analysis and Design with applications, Grady Booch et al, 3Rd Edition, Pearson, 2007
  - Web