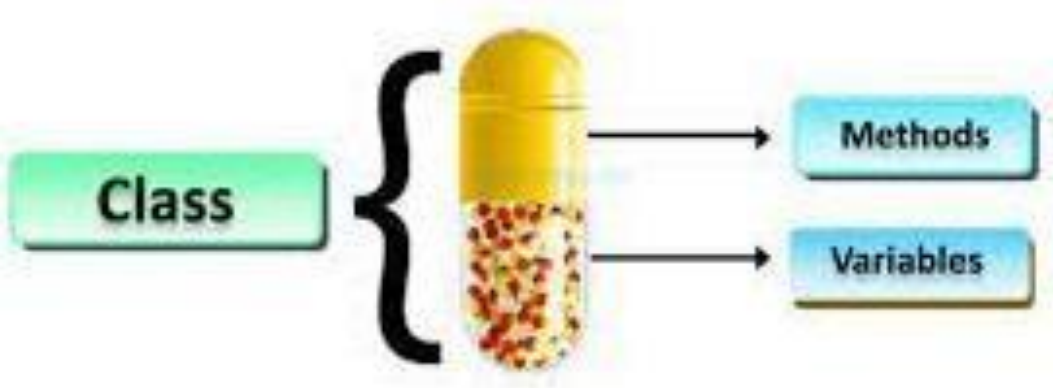
Encapsulation

- **Encapsulation is a fundamental principle in OOP that *binds* data (*attributes*) and the functions (*methods*) that operate on that data into a single unit called a *class*.**



Encapsulation in C++ with Example

```
#include <iostream>
using namespace std;
class Employee
{
        private:
        int amt;
public:
void setAmount(int a)
{
        amt=a;
}
int getAmount()
{
         return amt;
}
};
```

```
int main()
{
      Employee emp;
      emp.setAmount(1000);
      cout << emp.getAmount();
      return 0;
}
```

- **The main benefits of encapsulation is data protection.**
- ***As discussed earlier*, data protection protects data from unauthorized modification.**
- ***Also, as explained earlier, the first step to creating a class involves determining the attributes of an object, and subsequently dealing with the object as a whole.***

Class including function definition and function implementation

- **The following example illustrates how to declare a class that includes one function definition and implementation.**

```
// declaration section:
class Student
{
private:
    int idNum;
    string lastName;
    double gradeflointAverage;
public:
    void displayStudentData();
};
// implementation section:
void Student::displayStudentData()   //Note the using of ::
{
    cout << "Student #" << idNum << "'s last name is " <<
        lastName << endl;
    cout << "The grade point average for this student is " <<
        gradeflointAverage << endl;
}
```

The scope resolution operator (::) is used to define class methods outside the class body.

- **In the class definition shown in the above slide, the declaration section includes a function prototype, and the implementation section contains the actual function.**

- **The displayStudentData() function header is preceded by the class name, Student, and *the scope resolution operator* (::).**

- **You must use both the class name and the scope resolution operator when you implement a member function, because they tie the function to this class and allow every instantiated object to use the function name.**

- **The statements within the function are similar to those in any other function; in this case, they just display the object's data fields.**

Consider the following class definition:

```
class Student
{
        private:
           int idNum;
           string lastName;
           double gradePointAverage;
        public:
           void displayStudentData();
           void setIdNum(int);
           void setLastName(string);
           void setGradePointAverage(double);
};
```

- **You can place any lines of code you want within these functions when you implement them.**

- **If you want the setLastName() function to assign a string to the Student's lastName, then you can implement the function as follows:**

```
void Student::setLastName(string name)
{
    lastName = name;
}
```

- **The setLastName()function is a member of the Student class.**
- **You can determine this because the class header contains the Student class name and the scope resolution operator.**
- **The setLastName()function takes a string parameter that has the local identifier name.**
- **The function assigns the passed name value to the lastName field of Student object.**

- **When you write a program in which you instantiate a Student object named aJunior, you can assign a last name to the aJunior object with a statement such as the following:**

```
aJunior.setLastName("Basem");
```

- **The string "Basem" is passed into the aJunior object's setLastName()function, where it is copied to the aJunior object's lastName field.**

Consider the following example

```
#include<iostream>
#include<string>
using namespace std;
// declaration section
class Student
{
private:
    int idNum;
    string lastName;
    double gradeflointAverage;
public:
    void displayStudentData();
    void setIdNum(int);
    void setLastName(string);
    void setGradeflointAverage(double);
};
```

Continued

```
// implementation section
void Student::displayStudentData()
{
     cout << "Student #" <<idNum<<"'s last name is " <<
lastName << endl;
    cout << "The grade point average for this student is " <<
       gradeflointAverage << endl;
}
void Student::setId Num(int num)
{
     const int MAX_NUM = 9999;
     if (num <= MAX_NUM)
          id Num = num;
     else
          id Num = MAX_NUM;
}
```

Continued

```
void Student::setLastName(string name)
{
    lastName = name;
}

void Student::setGradeflointAverage(double gpa)
{
    const double MAX_GflA = 4.0;
    if (gpa <= MAX_GflA)
        gradeflointAverage = gpa;
    else
        gradeflointAverage = 0;
}
```

Continued

```
int main()
{
    Student aStudent;
    Student anotherStudent;
    aStudent.setLastName("Ahmed");
    aStudent.setIdNum(3456);
    aStudent.setGradeflointAverage(3.5);
    aStudent.displayStudentData();
    anotherStudent.setLast Name("Ibrahim");
    anotherStudent.setId Num(34567);
    anotherStudent.setGradeflointAverage(4.5);
    anotherStudent.displayStudentData();
    return 0;
}
```

Sample Output:

```
Student #3456's last name is Ahmed
The grade point average for this student is 3.5
Student #9999's last name is Ibrahim
The grade point average for this student is 0
```

- **Ex.2: The following program code contains uses a class named Carpet that contains data fields that store a Carpet's length, width, and price.**

- **Assume that a Carpet's price can be one of three values based on the Carpet's area.**

- **The Carpet class contains public functions that set the length and width, but because the price is determined by the length and width, it only has one public function that retrieves its value.**

- **The value of price is set by a private function that is called any time the length or width changes.**

- **The shaded setPrice() function is not intended to be used by a client function, such as the main()function.**
- ***Instead, setPrice() is used only by the two functions that set carpet dimensions.***
- **It makes sense that setPrice() should be private because you would not want a client program to change a Carpet's price to one that violated the carpet pricing rules.**

```
#include<iostream>
using namespace std;
// declaration section
class Carpet
{
private:
  int length;
  int width;
  double price;
  void setprice();
public:
  int getLength();
  int getWidth();
  double getprice();
  void setLength(int);
  void setWidth(int);
};
```

Continued:

```
// implementation section
int Carpet::getLength()
{
   return length;
}
int Carpet::getWidth()
{
   return width;
}
double Carpet::getprice()
{
   return price;
}
void Carpet::setLength(int len)
{
   length = len;
   setprice();
}
```

Continued:

```
void Carpet::setWidth(int wid)
{
     width = wid;
     setprice();
}
void Carpet::setprice()
{
     const int SMALL = 12;
     const int MED = 24;
     const double PRICE1 = 29.99;
     const double PRICE2 = 59.99;
     const double PRICE3 = 89.99;
     int area = length * width;
        if (area <= SMALL)
           price = PRICE1;
        else
        if (area <= MED)
           price = PRICE2;
        else
           price = PRICE3;
}
```

Continued:

```
int main()
{
    Carpet aRug;
    const char QUIT = 'Q';
    char dim;
    int length;
    int width;
    aRug.setLength(1);
    aRug.setWidth(1);
    cout << "Enter L to enter length or " <<
    "W to enter width or " <<
    QUIT << " to quit > ";
    cin >> dim;
```

Continued:

```
while (dim != QUIT)
{
  if (dim == 'L')
  {
     cout<< "Enter  a length > ";
     cin >> length;
     aRug.setLength(length);
     cout << "Lengthis  " << aRug.getLength()<<
     " Widthis " << aRug.getWidth() << endl  <<
     "price is " << aRug.getprice()  << endl;
  }
  else
  {
     cout<< "Enter  a width > ";
     cin >> width;
     aRug.setWidth(width);
     cout << "Lengthis  " << aRug.getLength()<<
     " Widthis " << aRug.getWidth() << endl  <<
     "price is " << aRug.getprice()  << endl;
  }
```

Continued:

```
 cout << "Enter L to enter length or W" <<
    "to enter width or " <<
     QUIT << " to quit > ";
    cin >> dim;
   }
   return 0;
}
```