Computer Programming (b) - E1124

(Spring 2021-2022)

Lecture 11

Encapsulation & Inheritance

INSTRUCTOR

Dr / Ayman Soliman

➢ Contents

- ➢ Encapsulation
- ➢ Access Private Members
- ➢ Why Encapsulation?
- ➢ C++ Inheritance
- ➢ Why and When to Use "Inheritance"?
- ➢ Multilevel Inheritance
- ➢ Access Specifiers

➢ Encapsulation

- The meaning of Encapsulation, is to make sure that "sensitive" data is hidden from users.

- To achieve this, you must declare class variables/attributes as private (cannot be accessed from outside the class).

- If you want others to read or modify the value of a private member, you can provide public get and set methods.

➢ Access Private Members

➢ To access a private attribute, use public "get" and "set" methods:

Example

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

class Employee {
  private:
    // Private attribute
    int salary;

  public:
    // Setter
    void setSalary(int s) {
      salary = s;
    }
    // Getter
    int getSalary() {
      return salary;
    }
};

int main() {
  Employee myObj;
  myObj.setSalary(50000);
  cout << myObj.getSalary();
  return 0;
}
```

```
50000
```

➢ Example explanation

- The salary attribute is private, which have restricted access.
- The public setSalary() method takes a parameter (s) and assigns it to the salary attribute (salary = s).
- The public getSalary() method returns the value of the private salary attribute.
- Inside main(), we create an object of the Employee class. Now we can use the setSalary() method to set the value of the private attribute to 50000. Then we call the getSalary() method on the object to return the value.

➢ Why Encapsulation?

- ➢ It is considered good practice to declare your class attributes as private (as often as you can).

- ➢ Encapsulation ensures better control of your data, because you (or others) can change one part of the code without affecting other parts

- ➢ Increased security of data

➢ C++ Inheritance

➢ In C++, it is possible to inherit attributes and methods from one class to another.

We group the "inheritance concept" into two categories:

- ❑ derived class (**child**) - the class that inherits from another class
- ❑ base class (**parent**) - the class being inherited from

➢ To inherit from a class, use the : symbol.

➢ In the next example, the Car class (child) inherits the attributes and methods from the Vehicle class (parent):

➢ Example

```
// Base class
class Vehicle {
  public:
    string brand = "Ford";
    void honk() {
      cout << "Tuut, tuut! \n" ;
    }
};

// Derived class
class Car: public Vehicle {
  public:
    string model = "Mustang";
};
```

```
  int main() {
Car myCar;
myCar.honk();
cout << myCar.brand + " " + myCar.model;
  return 0;
  }
```

```
Tuut, tuut!
Ford Mustang
```

➢ Why and When to Use "Inheritance"?

- ➢ It is useful for code reusability: reuse attributes and methods of an existing class when you create a new class.

➢ Multilevel Inheritance

- A class can also be derived from one class, which is already derived from another class.

- In the following example, MyGrandChild is derived from class MyChild (which is derived from MyClass).

➢ Example

```
// Base class (parent)
class MyClass {
  public:
    void myFunction() {
      cout << "Some content in parent
class." ;
    }
};

// Derived class (child)
class MyChild: public MyClass {
};

// Derived class (grandchild)
class MyGrandChild: public MyChild {
};
```

```
int main() {
  MyGrandChild myObj;
  myObj.myFunction();
  return 0;
}
```

```
Some content in parent class.
```

➢ Multiple Inheritance

➢ A class can also be derived from more than one base class, using a comma-separated list:

```
// Base class
class MyClass {
  public:
    void myFunction() {
      cout << "Some content in parent class." ;
    }
};

// Another base class
class MyOtherClass {
  public:
    void myOtherFunction() {
      cout << "Some content in another
class." ;
    }
};
```

```
// Derived class
class MyChildClass: public MyClass, public MyOt
herClass {
};

int main() {
  MyChildClass myObj;
  myObj.myFunction();
  myObj.myOtherFunction();
  return 0;
}
```

```
Some content in parent class.
Some content in another class.
```

➢ Access Specifiers

- You learned from the Access Specifiers types that there are three specifiers available in C++.

- Until now, we have only used public (members of a class are accessible from outside the class) and private (members can only be accessed within the class).

- The third specifier, protected, is similar to private, but it can also be accessed in the inherited class:

➢ Example

```
// Base class
class Employee {
  protected: // Protected access specifier
    int salary;
};

// Derived class
class Programmer: public Employee {
  public:
    int bonus;
    void setSalary(int s) {
      salary = s;
    }
    int getSalary() {
      return salary;
    }
};
```

```
int main() {
  Programmer myObj;
  myObj.setSalary(50000);
  myObj.bonus = 15000;
  cout << "Salary: " << myObj.getSalary() << "\n";
  cout << "Bonus: " << myObj.bonus << "\n";
  return 0;
}
```

```
Salary: 50000
Bonus: 15000
```

Thank you