OBJECT ORIENTED PROGRAMMING (OOP) WITH C++
INHERITANCE

AIUB. FALL 2017

Dr. Mahbubul Syeed
Associate Professor, Department. of CS, AIUB
mahbubul.syeed@aiub.edu
www.msyeed.weebly.com

CONTENTS

- ✓ Inheritance (What, why and how)
  - ✓ Base class and derived class in relation to inheritance (real life example)
  - ✓ What is inherited in inheritance
  - ✓ Access control in Inheritance (public, private and protected)
  - ✓ Multiple inheritances.

ACCESSING BASE CLASS **PUBLIC** VARIABLES AND METHODS

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

class Vehicle{
   string engine;
   double fuelLevel;
   public:
      string vechicleName;
      void setFuleLevel(double level){fuelLevel = level;}
      void setEngineConfig(string enginconf){engine = enginconf;}
      double getFuelAmount(){return fuelLevel;}
      string getEngineConfig(){return engine;}
   };
```

```
class Car: public Vehicle {
     int noOfDoors;
     public:
        Car(int doors, string name){
          noOfDoors = doors;
          vechicleName = name; // direct access to base class public variable
        }
        int getDoors (){return noOfDoors; }
 };

class Bike: public Vehicle {
    string bikeType;
     public:
        void setBikeType(string type){bikeType = type;}
        string getBikeType(){ return bikeType; }
  };
```

```
int main(){
        Car myCar(5, “Axio”);
        myCar.setEngineConfig("1050cc");
        myCar.setFuleLevel(50.6);

        cout << "My car config: " << endl<< myCar.getDoors() << endl<< myCar.getFuelAmount() << endl<< myCar.getEngineConfig();
}
```

ACCESSING BASE CLASS **PUBLIC** VARIABLES AND METHODS

Child / derived class **can directly access**:

- Base class / parent class public variables.
- Base class / parent class public functions.

ACCESSING BASE CLASS **PRIVATE** VARIABLES AND METHODS

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

class Vehicle{
   private:
      string engine;
      double fuelLevel;
   public:
      void setFuleLevel(double level){fuelLevel = level;}
      void setEngineConfig(string enginconf){engine = enginconf;}
      double getFuelAmount(){return fuelLevel;}
      string getEngineConfig(){return engine;}
};
```

```
class Car: public Vehicle {
   int noOfDoors;
   public:
      Car(int doors){ noOfDoors = doors; }

      void setCarDetail(){
         //trying to access base class private variable directly
         engine = "2000cc";

         //accessing private variable of base class through public function
         setEngineConfig("2000cc");
      }

      int getDoors (){return noOfDoors; }
};
```

```
int main(){

      Car myCar(5);

      myCar.setCarDetail();

      myCar.setFuleLevel(60.4);

      cout << "My car config: " << endl<< myCar.getDoors() << endl<< myCar.getFuelAmount() << endl<< myCar.getEngineConfig();

}
```

ACCESSING BASE CLASS **PRIVATE** VARIABLES AND METHODS

Child / derived class **can never directly access**:

- Base class / parent class private variables.
- Base class / parent class private functions.

Derived class can only access base class private data members through **base class public functions**.

ACCESSING BASE CLASS **PROTECTED** VARIABLES AND METHODS

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

class Vehicle{
    private:
        string engine;
    protected:
        double fuelLevel;
        void setEngineConfig(string enginconf){engine = enginconf;}
    public:
        void setFuleLevel(double level){fuelLevel = level;}
        double getFuelAmount(){return fuelLevel;}
        string getEngineConfig(){return engine;}
};
```

```
class Car: public Vehicle {
    int noOfDoors;
    public:
        Car(int doors){
            noOfDoors = doors;
            //accessing protected variable and function of super class directly from sub class
            fuelLevel = 40.5;
            setEngineConfig("1500cc");
        }
        int getDoors (){return noOfDoors; }
};
```

```
int main(){

Car myCar(5);

cout << "My car config: " << endl<< myCar.getDoors() << endl<< myCar.getFuelAmount() << endl<< myCar.getEngineConfig();

}
```

ACCESSING BASE CLASS **PROTECTED** VARIABLES AND METHODS

Child / derived class **can directly access**:

- Base class / parent class protected variables directly.
- Base class / parent class protected functions directly.

ACCESSING BASE CLASS **PROTECTED** VARIABLES AND METHODS

Protected variables can not be accessed outside the class with class object and dot operator!

This is same as private variable!!

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

class Vehicle{
    private:
        string engine;
    protected:
        double fuelLevel;
        void setEngineConfig(string enginconf){engine = enginconf;}
    public:
        void setFuleLevel(double level){fuelLevel = level;}
        double getFuelAmount(){return fuelLevel;}
        string getEngineConfig(){return engine;}
};
```

```
class Car: public Vehicle {
    int noOfDoors;
    public:
        Car(int doors){
            noOfDoors = doors;
            //accessing protected variable and function of super class directly from sub class
            fuelLevel = 40.5;
            setEngineConfig("1500cc");
        }
        int getDoors (){return noOfDoors; }
};
```

```
int main(){

    Car myCar(5);

    myCar.setEngineConfig("1500cc");  // cant access outside class with dot operator!!

    myCar.fuelLevel= 30.5;

    cout << "My car config: " << endl<< myCar.getDoors() << endl<< myCar.getFuelAmount() << endl<< myCar.getEngineConfig();

}
```

INHERITANCE TABLE

Access	public	protected	private
Same class	yes	yes	yes
Derived classes	yes	yes	no
Outside classes	yes	no	no

TYPES OF INHERITANCE

DERIVED CLASS INHERITS BASE CLASS AS **PUBLIC**

```
class DerivedClass : public BaseClass{
};
```

Base Class	*Inherited as*	Derived Class
Private		Private
Protected		Protected
Public		Public

DERIVED CLASS INHERITS BASE CLASS AS **PROTECTED**

class DerivedClass : ***protected*** *BaseClass{*
};

Base Class	*Inherited as*	Derived Class
Private		Private
Protected		Protected
Public		**Protected**

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

class Vehicle{
    private:
        string engine;
    protected:
        double fuelLevel;
        void setEngineConfig(string enginconf){engine = enginconf;}
    public:
        void setFuleLevel(double level){fuelLevel = level;}
        double getFuelAmount(){return fuelLevel;}
        string getEngineConfig(){return engine;}
};
```

```
class Car: protected Vehicle {
    int noOfDoors;
    public:
        Car(int doors){
            noOfDoors = doors;
            //accessing protected variable and function of super class directly from sub class
            fuelLevel = 40.5;
            setEngineConfig("1500cc");
        }
        int getDoors (){return noOfDoors; }
};
```

```
int main(){
    Car myCar(5);
    myCar.setEngineConfig("1500cc");  // cant access outside class with dot operator!!
    myCar.fuelLevel= 30.5;

    cout << "My car config: " << endl
        << myCar.getDoors() << endl
        << myCar.getFuelAmount() << endl
        << myCar.getEngineConfig();
}
```

CONSTRUCTOR AND DESTRUCTOR CALL SEQUENCE IN INHERITANCE

Constructor

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

class Base{
   public:
   int m_id;

   Base(int id=0)
   {
        m_id = id;
        cout << "Constructing Base\n";
    }
   int getId() { return m_id; }
};

class Derived: public Base{
    public:
    double m_cost;
    Derived(double cost=0.0)    {
        m_cost = cost;
        cout << "Constructing Derived\n";
    }
    double getCost() const { return m_cost; }
};
```

```
int main(){
   Derived cDerived;
    return 0;
}
```

In inheritance:
Creating derived class object always calls base class constructor first, then calls Its constructor (last)!

```
Constructing Base
Constructing Derived
```

Destructor

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

class Base{
   public:
   int m_id;

   Base(int id=0)
   {
        m_id = id;
        cout << "Constructing Base\n";
    }
   ~Base(){cout<<"Destructing base \n";}
   int getId() { return m_id; }
};

class Derived: public Base{
    public:
    double m_cost;
    Derived(double cost=0.0)    {
        m_cost = cost;
        cout << "Constructing Derived\n";
    }
     ~Derived(){cout<<"Destructing derived \n";}
    double getCost() const { return m_cost; }
};
```

```
int main(){
   Derived cDerived;
    return 0;
}
```

In inheritance:
Destructor is called exactly the reverse of the constructor call!!

```
Constructing Base
Constructing Derived
Destructing derived
Destructing base
```

```
#include<iostream>
using namespace std;
class A
{
public:
A() {  cout << " constructing A" << endl; }
~A() {  cout << " destructing A" << endl; }
};

class B: public A
{
public:
B() { cout << " constructing B" << endl; }
~B() {  cout << " destructing B" << endl; }
};

class C: public B
{
public:
C(){ cout << " constructing C" << endl; }
~C() {  cout << " destructing C" << endl; }
};

class D: public C
{
public:
D() { cout << " constructing D" << endl; }
~D() {  cout << " destructing D" << endl; }
};
```

What is the output??

```
int main()
{
   cout << "Constructing A: " << endl;
   A cA;

   cout << "Constructing B: " << endl;
   B cB;

   cout << "Constructing C: " << endl;
   C cC;

   cout << "Constructing D: " << endl;
   D cD;
}
```

```
Constructing A:
  constructing A
Constructing B:
  constructing A
  constructing B
Constructing C:
  constructing A
  constructing B
  constructing C
Constructing D:
  constructing A
  constructing B
  constructing C
  constructing D
```

```
destructing D
destructing C
destructing B
destructing A
destructing C
destructing B
destructing A
destructing B
destructing A
destructing A
```