Lab Work 4

Learning Objectives

- What is inheritance?
- Protected member and class access
- Constructors and destructors in base and derived classes
- Class hierarchies
- Polymorphism and virtual member functions
- Abstract base classes
- Pure virtual functions
- Multiple inheritance

Inheritance: Code Drill 1

a) Please run these code in Dev C++

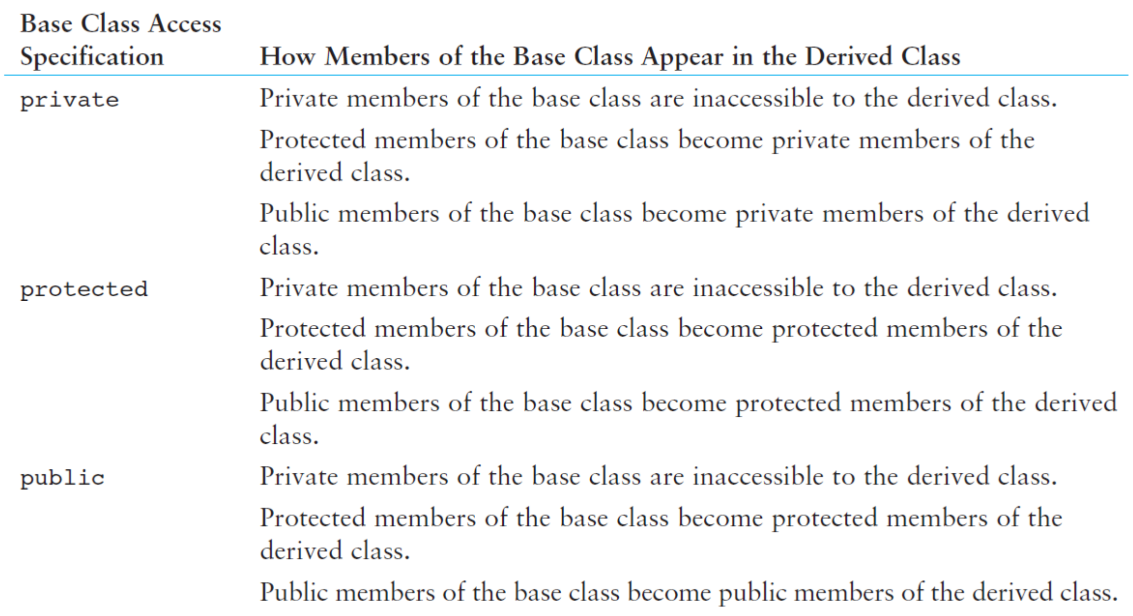
```
#include <iostream>
using namespace std;
//*******************************
// BaseClass declaration        *
//*******************************
class BaseClass
{
public:
   BaseClass()  // Constructor
      { cout << "This is the BaseClass constructor.\n"; }
   ~BaseClass() // Destructor
      { cout << "This is the BaseClass destructor.\n"; }
};
//*******************************
// DerivedClass declaration      *
//*******************************

class DerivedClass : public BaseClass
{
public:
   DerivedClass()  // Constructor
      { 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;
}
```



Base Class Access Specification	How Members of the Base Class Appear in the Derived Class
`private`	Private members of the base class are inaccessible to the derived class.
	Protected members of the base class become private members of the derived class.
	Public members of the base class become private members of the derived class.
`protected`	Private members of the base class are inaccessible to the derived class.
	Protected members of the base class become protected members of the derived class.
	Public members of the base class become protected members of the derived class.
`public`	Private members of the base class are inaccessible to the derived class.
	Protected members of the base class become protected members of the derived class.
	Public members of the base class become public members of the derived class.

What about static member variables ? : Discussion

File Name= Tree.h

```
// Tree class
class Tree
{
private:
   static int objectCount;    // Static member variable.
public:
   // Constructor
   Tree()
      { objectCount++; }

   // Accessor function for objectCount
   int getObjectCount() const
      { return objectCount; }
};

// Definition of the static member variable, written
// outside the class.
int Tree::objectCount = 0;
```

File name =TreeTest.cpp

```
#include <iostream>
#include "Tree.h"
using namespace std;

int main()
{
   // Define three Tree objects.
   Tree oak;
   Tree elm;
   Tree pine;

   // Display the number of Tree objects we have.
   cout << "We have " << pine.getObjectCount()
        << " trees in our program!\n";
   return 0;
}
```

Try to create a new project called Tree in your DevC++. Then add two files : Tree.h and TreeTest.cpp. Try copy paste the output of this codes here.

Output (please paste below)

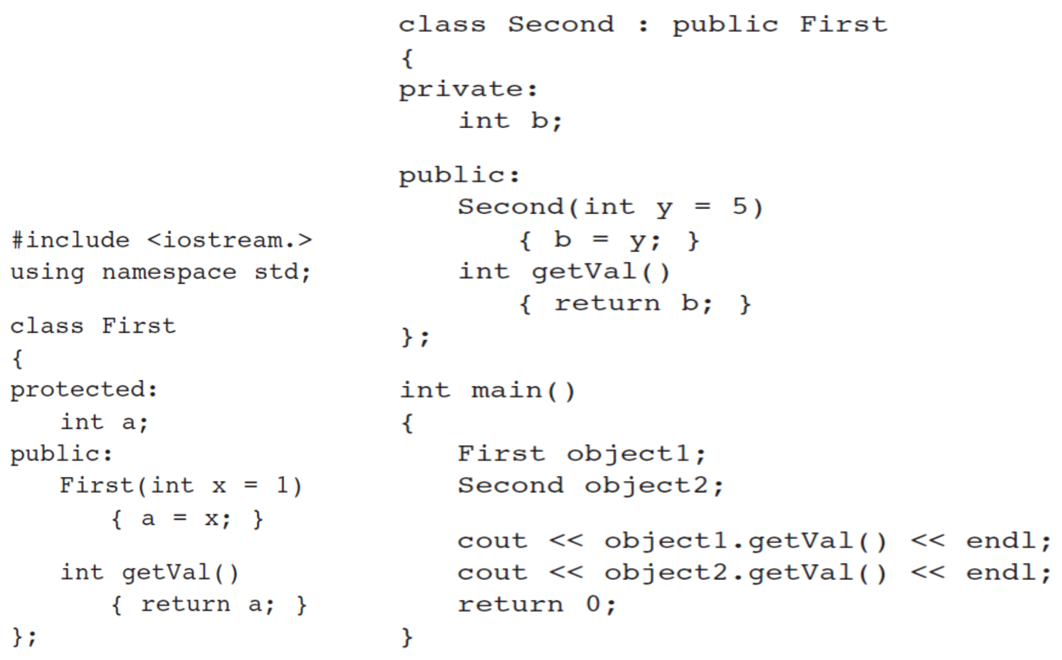
//paste here

Discussion on the codes (please write out your thoughts)

- Your thoughts here
- //more thoughts

Question 1: Inheritance

Please take a look at the code below and answer questions (a) to (e) below:



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

class First
{
protected:
   int a;
public:
   First(int x = 1)
      { a = x; }

   int getVal()
      { return a; }
};
```

```
class Second : public First
{
private:
   int b;

public:
   Second(int y = 5)
      { b = y; }
   int getVal()
      { return b; }
};

int main()
{
   First object1;
   Second object2;

   cout << object1.getVal() << endl;
   cout << object2.getVal() << endl;
   return 0;
}
```

a) Read the code properly and identify which one is the derived class and base class.
b) After that, please draw out a UML class diagram that can represent the relationship between the two classes shown above.
c) Study the code carefully and write out the output for the program.
d) Use a yellow highlighter to identify the constructor for both the classes.
e) Use the blue highlighter to identify the protected, public, private members of the classes.
f) Explain how the codes will be impacted with the words highlighted in (e).

Multiple inheritance: Discussion

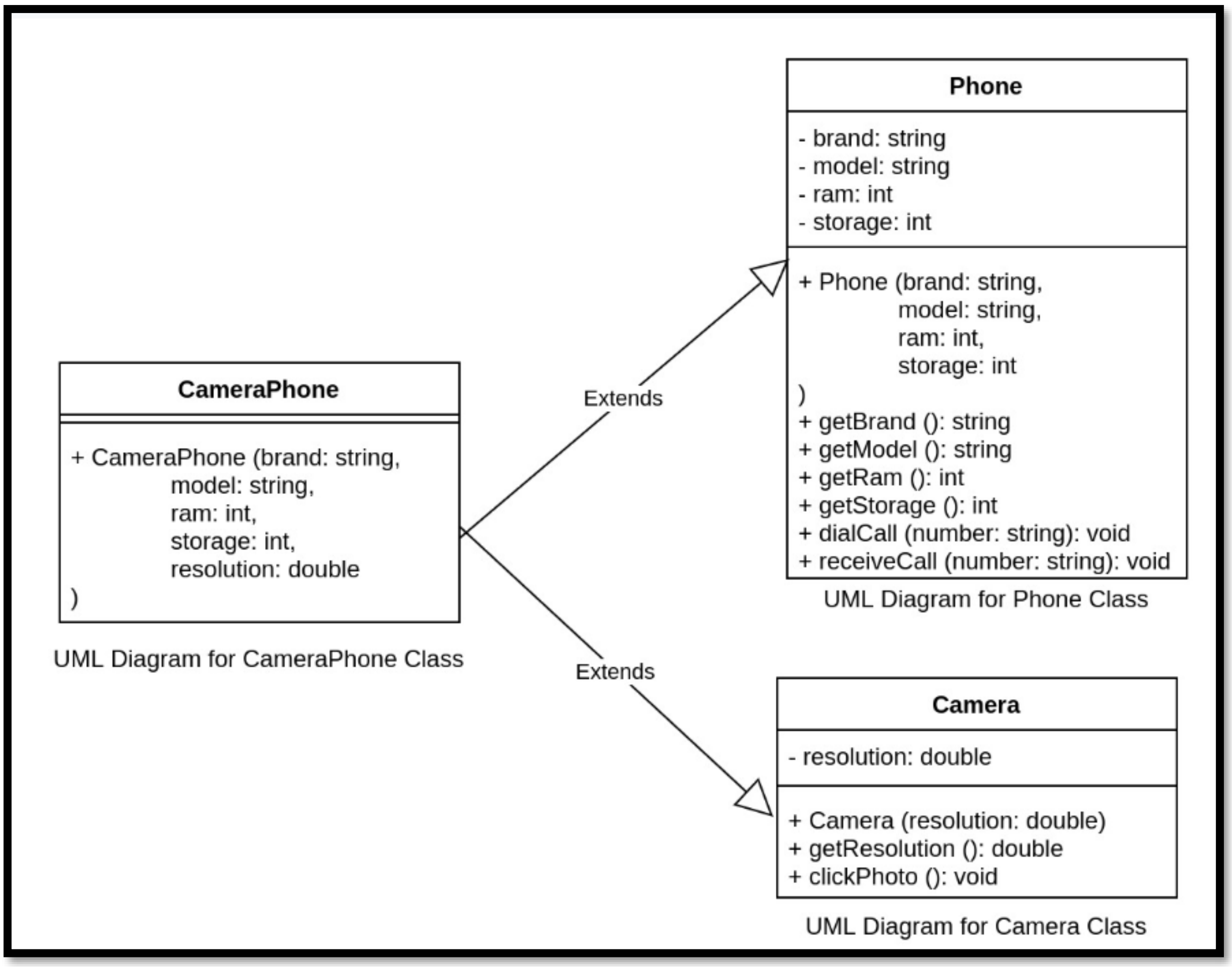

Phone
- brand: string
- model: string
- ram: int
- storage: int
+ Phone (brand: string, model: string, ram: int, storage: int)
+ getBrand (): string
+ getModel (): string
+ getRam (): int
+ getStorage (): int
+ dialCall (number: string): void
+ receiveCall (number: string): void
UML Diagram for Phone Class
CameraPhone
+ CameraPhone (brand: string, model: string, ram: int, storage: int, resolution: double)
UML Diagram for CameraPhone Class
Extends
Extends
Camera
- resolution: double
+ Camera (resolution: double)
+ getResolution (): double
+ clickPhoto (): void
UML Diagram for Camera Class

Figure 1: Multiple inheritance

The code will look something like this

```
class Phone {
//code here
};

class Camera {
 //code here
};

class CameraPhone: public Phone, public Camera {
public:
	CameraPhone(string brand, string model, int ram, int storage, double resolution): Phone(brand, model, ram, storage), Camera(resolution) {
	}
};
```

Here we mention the base classes as comma-separated-values with the inheritance-access-specifier. The constructors are also mentioned comma-separated with the required parameters.

In this particular example, CameraPhone does not have any property of its own and so we've not mentioned anything inside the constructor method. If there would have been a property in CameraPhone which was not inherited from its base classes, we can initialize it in the constructor just like we have been doing previously.

Homework

Study the main function below and write appropriate codes for the three classes show in Figure 1.

```
using namespace std;

int main() {
	// do not modify the main method
	Phone phone("Apple", "iPhone", 4, 64);
	Camera camera(24.1);
	CameraPhone cameraPhone("Apple", "iPhone 8", 4, 64, 12);

	cout << phone.getBrand() << " " << phone.getModel() << " " << phone.getRam() << " " << phone.getStorage() << endl;
	phone.dialCall("9732130450");
	phone.receiveCall("9732130450");

	cout << camera.getResolution() << endl;
	camera.clickPhoto();

	cout << cameraPhone.getBrand() << " " << cameraPhone.getModel() << " " << cameraPhone.getRam() << " " << cameraPhone.getStorage() << " " << cameraPhone.getResolution() << endl;
	cameraPhone.dialCall("9732130450");
	cameraPhone.receiveCall("9732130450");
```

```
        cameraPhone.clickPhoto();
        return 0;
}
```

Multi-level inheritance: Discussion

In Single Inheritance, there is a derived class which extends a base class whereas the base class is standalone and does not extend anything.Similar to Single Inheritance, we can have Multi-level Inheritance as well where the base class extends some other base class.

Example

Every Phone is-a ElectronicDevice

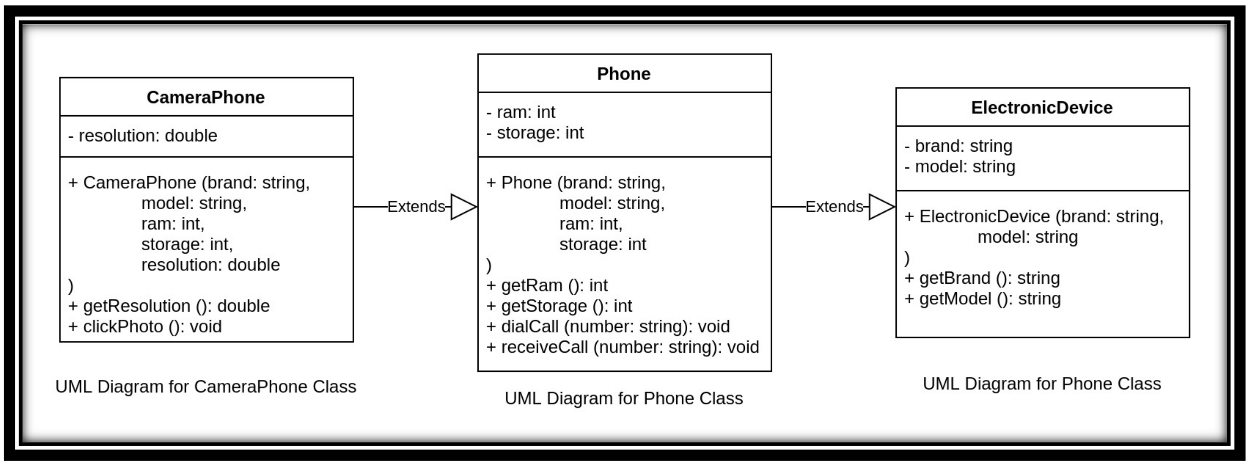
Every CameraPhone is-a Phone

Assuming that brand and model are properties of ElectronicDevice.

This can be represented as:

Here Phone extends ElectronicDevice and CameraPhone extends Phone.

The code would look something like this:



CameraPhone
- resolution: double
+ CameraPhone (brand: string, model: string, ram: int, storage: int, resolution: double)
+ getResolution (): double
+ clickPhoto (): void
Extends
Phone
- ram: int
- storage: int
+ Phone (brand: string, model: string, ram: int, storage: int)
+ getRam (): int
+ getStorage (): int
+ dialCall (number: string): void
+ receiveCall (number: string): void
Extends
ElectronicDevice
- brand: string
- model: string
+ ElectronicDevice (brand: string, model: string)
+ getBrand (): string
+ getModel (): string
UML Diagram for CameraPhone Class
UML Diagram for Phone Class
UML Diagram for Phone Class

Figure 2: Mult-level inheritance

The code will look somewhat like this

```
class ElectronicDevice {
 //code here
};

class Phone: public ElectronicDevice {
  //code here
};
class CameraPhone: public Phone {
  //code here
};
```

Inheritance Access Specifier: Discussion

While learning inheritance, we have been using public before the name of the base class while extending it like this:

class DerivedClass: public BaseClass

Here, public is an inheritance access specifier. By using public inheritance access specifier, we are able to maintain the original access specifier of the members of the base class in the derived class.

Let's see what will happen if we use protected inheritance access specifier like this:

class ProtectedDerivedClass: protected BaseClass

If we do this then all the public and protected class members of the BaseClass will become protected members of the DerivedClass instead of the original access specificity.

Let's see what will happen if we use private inheritance access specifier like this:

class PrivateDerivedClass: private BaseClass

If we do this then all the public and protected class members of the BaseClass will become private members of the DerivedClass instead of the original access specificity.

Note that the private members won't be accessible in the derived class irrespective of the inheritance access specificity**.**