

---

---

# Week 7: Inheritance

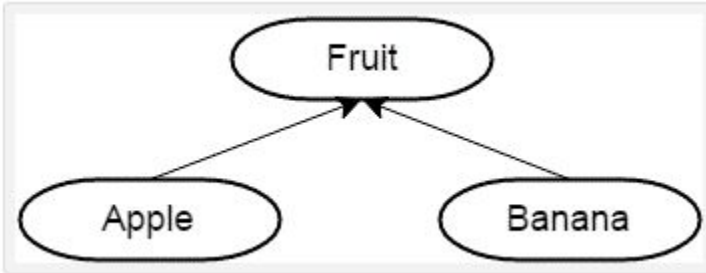
By, Mahrukh Khan

---

---

# Inheritance

- Inheritance is the action of succession of characteristics to child given by parents.
- Inheritance in OOP is the capability of one class to acquire properties and characteristics from another class



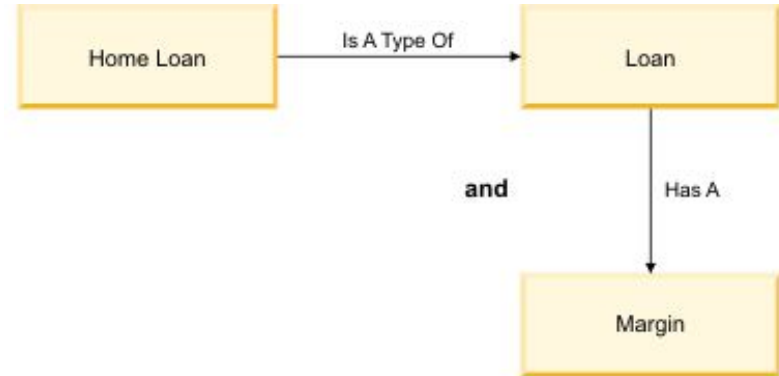
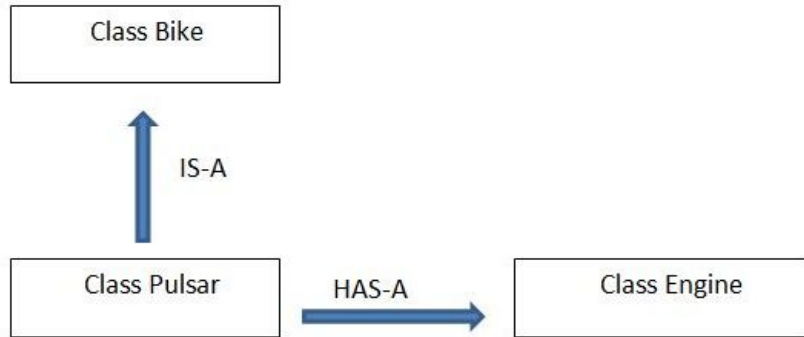
# Inheritance in OOP

- Inheritance is a mechanism of reusing and extending existing classes without modifying them, thus producing hierarchical relationships between them.
- it allows the child class to acquire the properties (the data members) and functionality (the member functions) of parent class.
- In an inheritance (is-a) relationship, the class being inherited from is called the parent class, base class, or superclass, and the class doing the inheriting is called the child class, derived class, or subclass.

```
class parent_class
{
    //Body of parent class
};
class child_class : access_modifier parent_class
{
    //Body of child class
};
```

# Is-a/Has-a Relationship

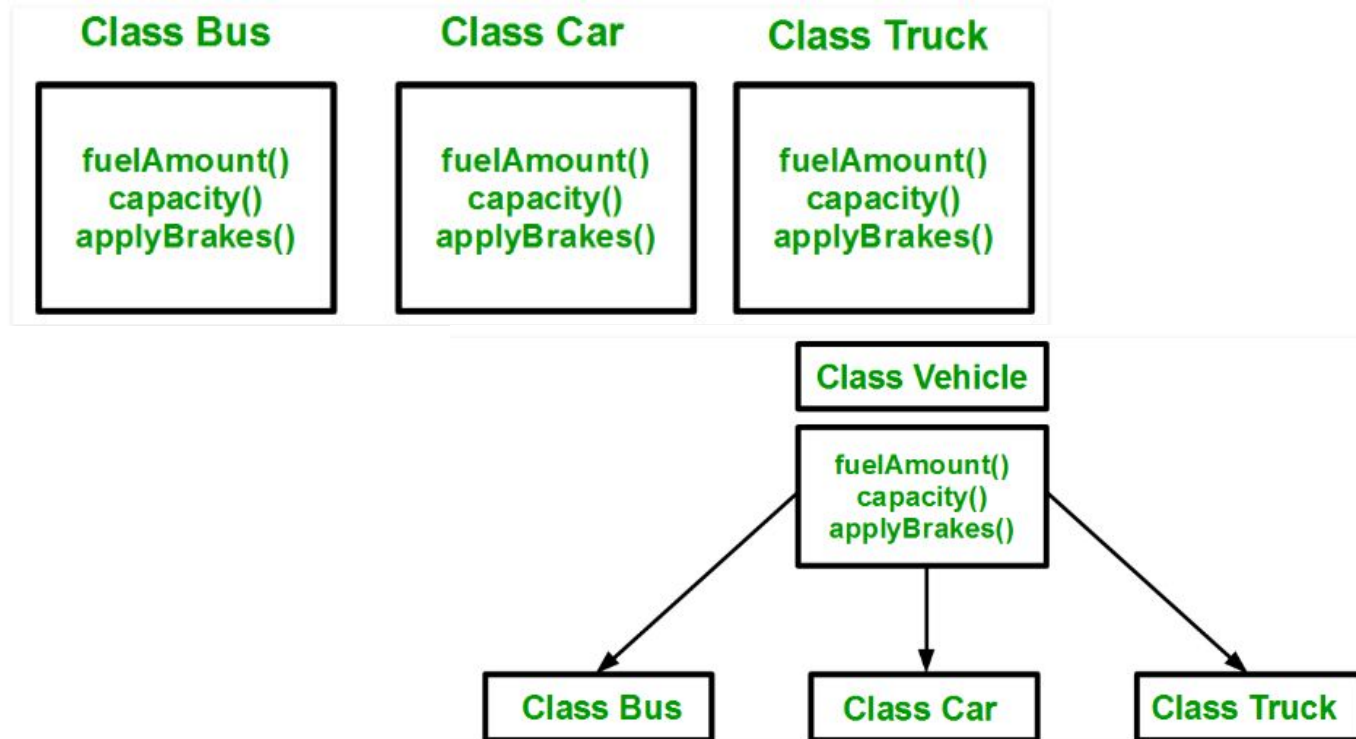
- **is-a relationship** is a totally based on Inheritance. For example, Apple is a Fruit, Car is a Vehicle etc.  
Inheritance is unidirectional. For example, House is a Building. But Building is not a House.
- **has-a relationship** in an object is called a member field of an object.



# Advantages

- **code reusability** and **readability**. When child class inherits the properties and functionality of parent class, we need not to write the same code again in child class. This makes it easier to reuse the code, makes us write the less code and the code becomes much more readable.
- **Less Execution Time/Less Memory**
- **Data Hiding**. Base class can decide to keep some data private so that it can not be altered by the derived class.

# Example-Reuse

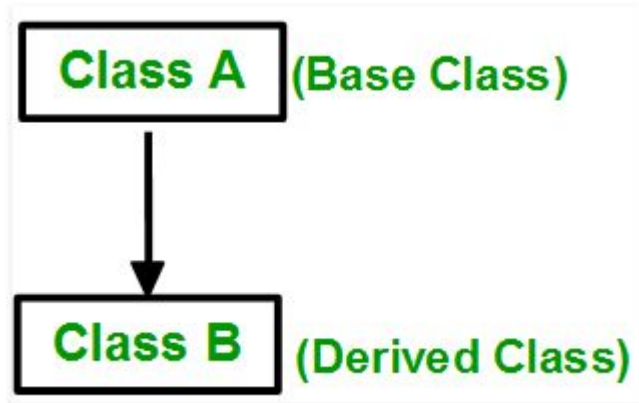


# Inheritance Types

- Single
- Multi Level
- Multiple
- Hierarchical
- Hybrid

# single

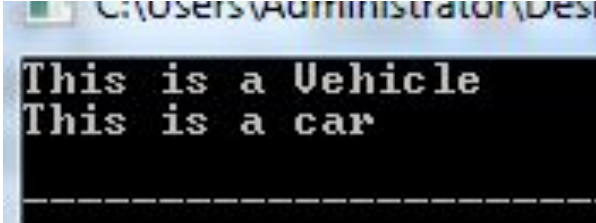
In Single inheritance one class inherits one class exactly.



```
class Vehicle {
public:
    Vehicle()
    {
        cout << "This is a Vehicle" << endl;
    }
};

class Car: public Vehicle{
public:
    Car()
    {
        cout<<"This is a car"<<endl;
    }
};

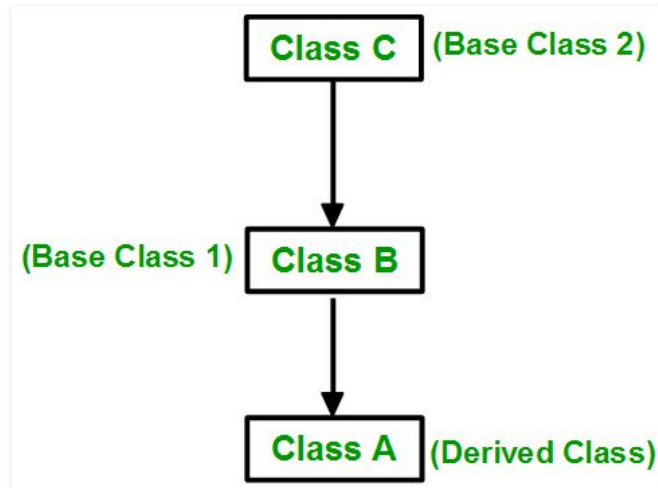
int main()
{
    Car obj;
    return 0;
}
```





# multi-level

In this type of inheritance one class inherits another child class.

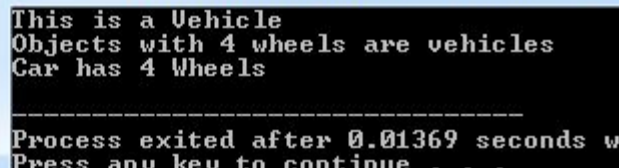


```
class Vehicle {
public:
    Vehicle()
    {
        cout << "This is a Vehicle" << endl;
    }
};

class fourWheeler: public Vehicle
{
public:
    fourWheeler()
    {
        cout<<"Objects with 4 wheels are vehicles"<<endl;
    }
};

class Car: public fourWheeler{
public:
    Car()
    {
        cout<<"Car has 4 Wheels"<<endl;
    }
};

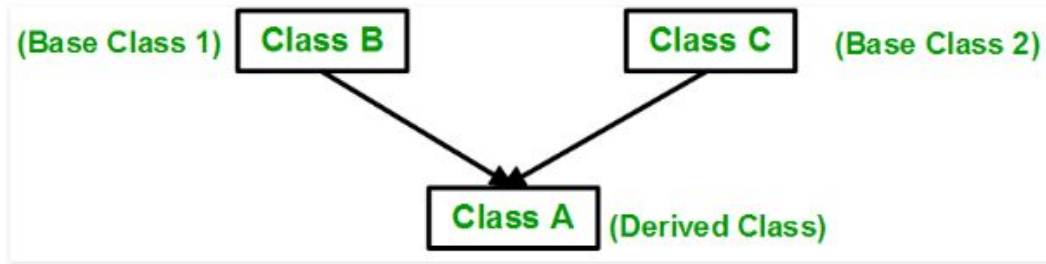
int main()
{
    Car obj;
    return 0;
}
```



```
C:\Users\Administrator\Desktop\CF-Spring2015\Less
This is a Vehicle
Objects with 4 wheels are vehicles
Car has 4 Wheels
-----
Process exited after 0.01369 seconds w
Press any key to continue . . .
```

# multiple

In multiple inheritance, a class can inherit from more than one class.



```

class Vehicle {
public:
    Vehicle()
    {
        cout << "This is a Vehicle" << endl;
    }
};

class FourWheeler {
public:
    FourWheeler()
    {
        cout << "This is a 4 wheeler Vehicle" << endl;
    }
};

class Car: public Vehicle, public FourWheeler{
public:
    Car()
    {
        cout<<"This is a car"<<endl;
    }
};

```

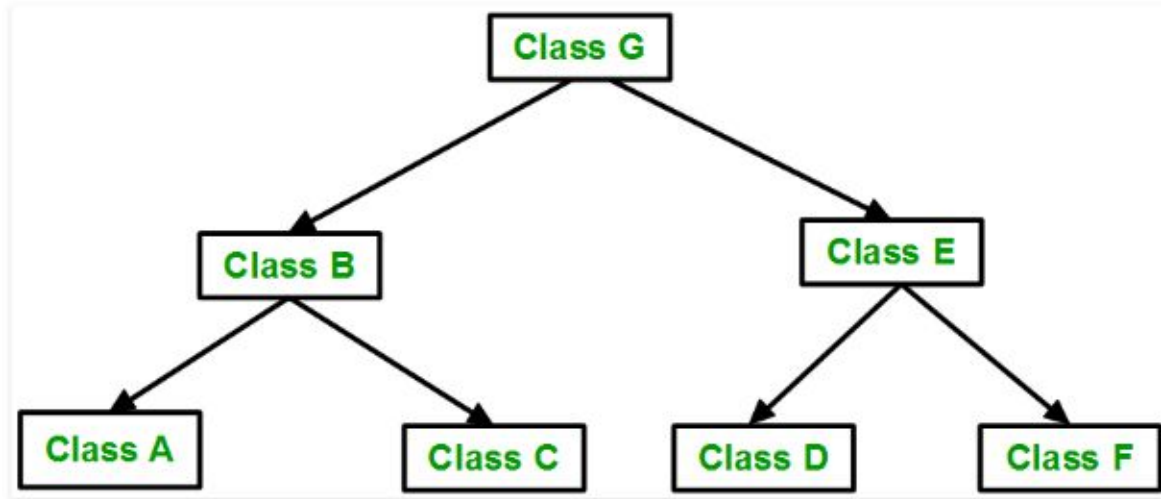
```

This is a Vehicle
This is a 4 wheeler Vehicle
This is a car
-----

```

# Hierarchical

In this type of inheritance, one parent class has more than one child class.



```

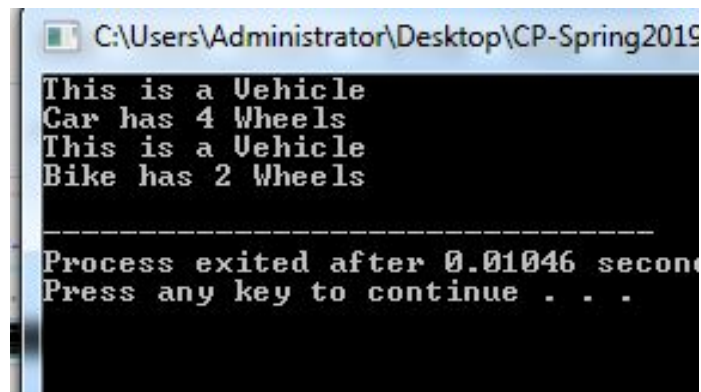
class Vehicle {
public:
    Vehicle()
    {
        cout << "This is a Vehicle" << endl;
    }
};

class Car: public Vehicle{
public:
    Car()
    {
        cout<<"Car has 4 Wheels"<<endl;
    }
};

class Bike : public Vehicle{
public:
    Bike()
    {
        cout<<"Bike has 2 Wheels"<<endl;
    }
};

int main()
{
    Car obj1;
    Bike obj2;
    return 0;
}

```



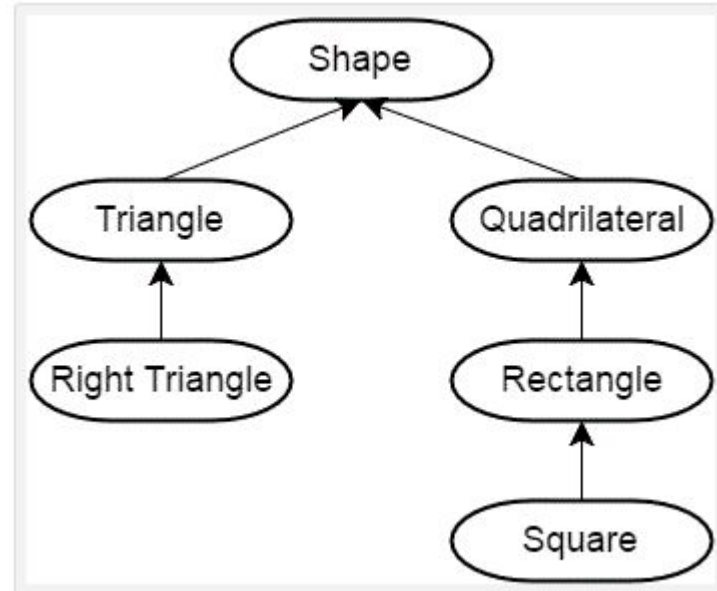
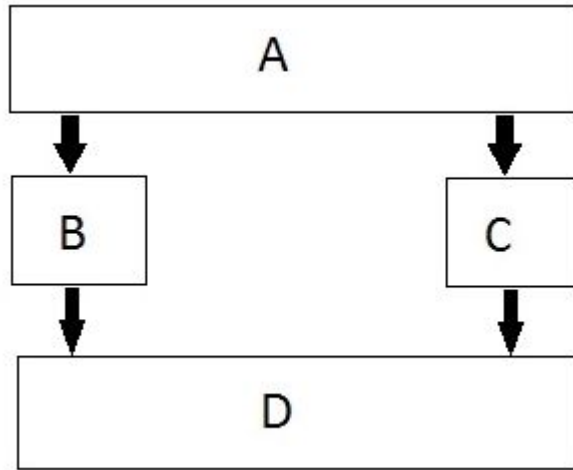
```

C:\Users\Administrator\Desktop\CP-Spring2019
This is a Vehicle
Car has 4 Wheels
This is a Vehicle
Bike has 2 Wheels
-----
Process exited after 0.01046 seconds
Press any key to continue . . .

```

# Hybrid

Hybrid inheritance is a combination of more than one type of inheritance. For example, A child and parent class relationship that follows multiple and hierarchical inheritance both can be called hybrid inheritance.



# Access Control Inheritance

When creating a derived class from a base class, you can use different access specifiers to inherit the data members of the base class. These can be public, protected or private.

- **Public mode:** If we derive a subclass from a public base class. Then the public member of the base class will become public in the derived class and protected members of the base class will become protected in derived class.
- **Protected mode:** If we derive a subclass from a Protected base class. Then both public member and protected members of the base class will become protected in derived class.
- **Private mode:** If we derive a subclass from a Private base class. Then both public member and protected members of the base class will become Private in derived class.



# Access Control Inheritance

Base class member access specifier	Type of Inheritance		
	Public	Protected	Private
Public	Public	Protected	Private
Protected	Protected	Protected	Private
Private	Not accessible (Hidden)	Not accessible (Hidden)	Not accessible (Hidden)

# Example

```
class A
{
public:
    int x;
protected:
    int y;
private:
    int z;
};
```

```
class B : public A
{
    // x is public
    // y is protected
    // z is not accessible from B
};
```

```
class C : protected A
{
    // x is protected
    // y is protected
    // z is not accessible from C
};
```

```
class D : private A    // 'private' is default for classes
{
    // x is private
    // y is private
    // z is not accessible from D
};
```

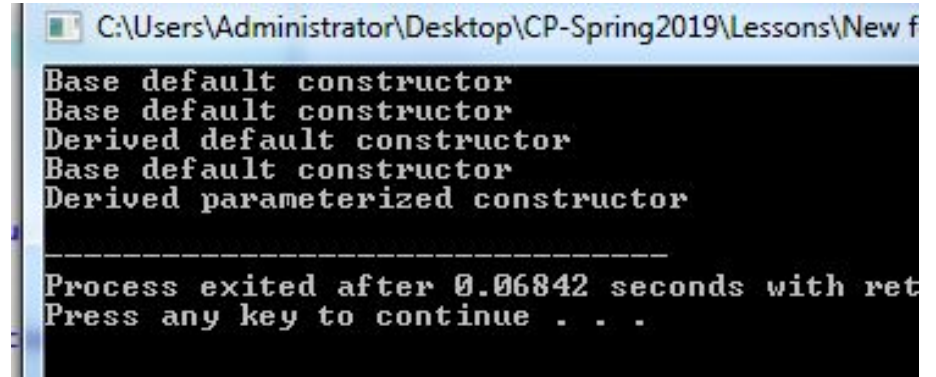
Base-class member-access specifier	Type of inheritance		
	public inheritance	protected inheritance	private inheritance
public	<p><b>public</b> in derived class.</p> <p>Can be accessed directly by member functions, <b>friend</b> functions and nonmember functions.</p>	<p><b>protected</b> in derived class.</p> <p>Can be accessed directly by member functions and <b>friend</b> functions.</p>	<p><b>private</b> in derived class.</p> <p>Can be accessed directly by member functions and <b>friend</b> functions.</p>
protected	<p><b>protected</b> in derived class.</p> <p>Can be accessed directly by member functions and <b>friend</b> functions.</p>	<p><b>protected</b> in derived class.</p> <p>Can be accessed directly by member functions and <b>friend</b> functions.</p>	<p><b>private</b> in derived class.</p> <p>Can be accessed directly by member functions and <b>friend</b> functions.</p>
private	<p>Hidden in derived class.</p> <p>Can be accessed by member functions and <b>friend</b> functions through <b>public</b> or <b>protected</b> member functions of the base class.</p>	<p>Hidden in derived class.</p> <p>Can be accessed by member functions and <b>friend</b> functions through <b>public</b> or <b>protected</b> member functions of the base class.</p>	<p>Hidden in derived class.</p> <p>Can be accessed by member functions and <b>friend</b> functions through <b>public</b> or <b>protected</b> member functions of the base class.</p>

Whether derived class's default constructor is called or parameterised is called, base class's default constructor is always called inside them

```
class Base
{
    int x;
public:
    Base() {
        cout << "Base default constructor\n";
    }
};

class Derived : public Base
{
    int y;
public:
    Derived() {
        cout << "Derived default constructor\n";
    }
    Derived(int i) {
        cout << "Derived parameterized constructor\n";
    }
};

int main()
{
    Base b;
    Derived d1;
    Derived d2(10);
}
```



A screenshot of a Windows command prompt window showing the output of a C++ program. The window title is "C:\Users\Administrator\Desktop\CP-Spring2019\Lessons\New f". The output text is as follows:

```
Base default constructor
Base default constructor
Derived default constructor
Base default constructor
Derived parameterized constructor

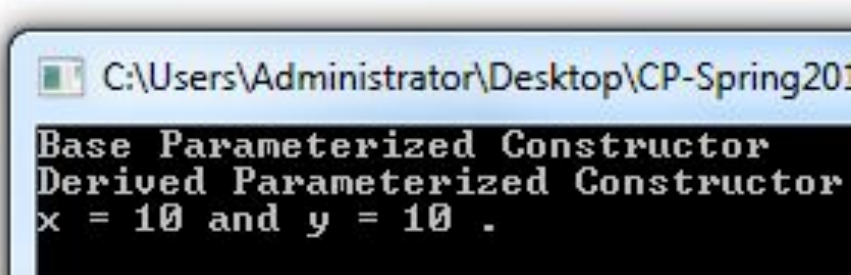
-----
Process exited after 0.06842 seconds with return code 0
Press any key to continue . . .
```

# Base class Parameterized Constructor in Derived Class Constructor

```
class Base
{
    int x;
public:
    Base() {
        cout << "Base default constructor\n";
    }
    Base(int i) { |
        x = i;
        cout << "Base Parameterized Constructor\n";
    }
    int getX() {
        return x;
    }
};
```

```
int main()
{
    Derived d2(10);
    d2.display();
}
```

```
class Derived : public Base
{
    int y;
public:
    Derived() {
        cout << "Derived default constructor\n";
    }
    Derived(int j):Base(j) {
        y = j;
        cout << "Derived Parameterized Constructor\n";
    }
    void display() {
        cout << "x = " << getX() << " and y = " << y << " . " << endl;
    }
};
```



```
C:\Users\Administrator\Desktop\CP-Spring201...
Base Parameterized Constructor
Derived Parameterized Constructor
x = 10 and y = 10 .
```

# Example

```
class Base
{
    int x;
public:
    Base() {
        cout << "Base default constructor\n";
    }
    Base(int j) {
        x = j;
        cout << "Base Parameterized Constructor\n";
    }
    int getX() {
        return x;
    }
};

class Derived : public Base
{
    int y;
public:
    Derived() {
        cout << "Derived default constructor\n";
    }
    Derived(int i, int j):Base(j) {
        y = i;
        cout << "Derived Parameterized Constructor\n";
    }
    void display() {
        cout << "x = " << getX() << " and y = " << y << " .\n";
    }
};
```

```
int main()
{
    Derived d2(10, 12);
    d2.display();
}
```

C:\Users\mahrukh.khan\Desktop\CP-Spring2019\Lessons\W

```
Base Parameterized Constructor
Derived Parameterized Constructor
x = 12 and y = 10 .
```

# Order of Destructor

```
class Vehicle {  
private:  
    int a;  
  
public:  
    Vehicle() {  
        cout << "This is a Vehicle" << endl;  
    }  
    ~Vehicle() {  
        cout << "Destructor for Vehicle" << endl;  
    }  
};
```

```
class Car: public Vehicle {  
public:  
    Car() {  
        cout<<"Car has 4 Wheels"<<endl;  
    }  
    ~Car() {  
        cout<<"Destructor for Car"<<endl;  
    }  
};
```

```
int main()  
{  
    Car obj1;  
    //Bike obj2;  
    return 0;  
}
```

C:\Users\mahrukh.khan\Desktop\CP-Spring2019\Lesson

```
This is a Vehicle  
Car has 4 Wheels  
Destructor for Car  
Destructor for Vehicle
```



# Exercise

Implement an inheritance hierarchy for students at a university. Use `Student` as the base class of the hierarchy, then include classes `UndergraduateStudent` and `GraduateStudent` that derive from `Student`. Continue to extend the hierarchy as deep (i.e., as many levels) as possible. For example, `Freshman`, `Sophomore`, `Junior` and `Senior` might derive from `UndergraduateStudent`, and `DoctoralStudent` and `MastersStudent` might derive from `GraduateStudent`.