**Object-Oriented Programming - Principles in C++:**

## Introduction

Let's start to learn object-oriented programming principles/features in C++.

This course is aimed at providing the OOP features like Inheritance and Abstraction. Encapsulation and Abstraction have been covered in the [OOP Basics course](https://workat.tech/courses/object-oriented-programming-basics-c++-tfkx6n7p9r9j).

Once you complete this course, you will be an expert in OOP principles. It is recommended that you try out a few [machine coding/low-level design problems](https://workat.tech/machine-coding/article/how-to-practice-for-machine-coding-kp0oj3sw2jca) after completing this course.

The course is supposed to be interactive as in you code and attempt quizzes as soon as you learn a new concept. This will help you get a better understanding of the concepts and will make it easier for you to become a better programmer.

There are multiple checkpoints where you are supposed to solve one or more assessments to test your understanding of the concepts learnt in the previous chapters. Submitting all the assessments correctly is mandatory for completing the course.

## Table of Contents

### *Inheritance*: Intro, Hierarchical Inheritence, Multiple Inheritance, Multilevel Inheritence, Hybdid Inheritence, protected, Inheritence Access Specifier

### Assessment – I: EMS, EMS-II, Assessment -I

### Polymorphism – I: Function Overloading, Operator Overloading – I, Compile-time Polymorphism, Function Hiding, EMS-III

### Assessment - II, III & IV

### Polymorphism – II: Function Template, Class Template

### Assessment - V

### Polymorphism – III: Pointer – to – derived class – I, II , Virtual Function, Runtime Polymorphism, Abstract Classes

### Assessment - VI & VII

### Additional Concepts: Composition, FYI, Destructor, Friend Function, Operator Overloading – II, III

**Inheritance**

**Introduction**

There are multiple classes that a phone can belong to. We can define multiple classes based on an hierarchy. Let's look at a few dependencies based on hierarchy.

* Every Human is-a Mammal
* Every Mammal is-a Organism
* Every Mammal is not a Human
* Every Organism is not a Mammal
* Every Phone is-a ElectronicDevice
* Every CameraPhone is-a Phone
* Every ElectronicDevice is not a Phone
* Every Phone is not a CameraPhone

As we can see that any classification that is below in the hierarchy has an "is-a" relationship with its ancestor.

In this relationship, the ancestor is known as the base class and the descendant is known as the derived class. In case of Phone and CameraPhone (which is-a Phone), **Phone is a base class and CameraPhone is the derived class**.

If we were to create classes for both, we will see that all the attributes and functions provided by Phone will also be provided by CameraPhone.

##### Example

Phone will have brand, model, ram, storage

CameraPhone will also have brand, model, ram, storage

CameraPhone will have additional attributes and will provide additional functionalities on top of what the Phone class provides.

***OOP languages allow us to reuse this and inherit attributes and functions from one class to another. This property is known as Inheritance.***

Here, CameraPhone = Phone + extra attributes + extra functions

We can create the Phone and CameraPhone classes like this:

class Phone {

private:

string brand;

string model;

int ram;

int storage;

public:

Phone (string brand, string model, int ram, int storage) {

this->brand = brand;

this->model = model;

this->ram = ram;

this->storage = storage;

}

string getBrand() {

return this->brand;

}

string getModel() {

return this->model;

}

int getRam() {

return this->ram;

}

int getStorage() {

return this->storage;

}

void dialCall (string number) {

cout << "Calling " << number << " from " << brand << ":" << model << "\n";

}

void receiveCall (string number) {

cout << "Receiving call from " << number << " on " << brand << ":" << model << "\n";

}

};

**class CameraPhone: public Phone {**

**double resolution;**

**public:**

**CameraPhone(string brand, string model, int ram, int storage, double resolution): Phone(brand, model, ram, storage) {**

**this->resolution = resolution;**

**}**

double getResolution() {

return this->resolution;

}

void clickPhoto () {

cout << "Clicking photo on a " << resolution << " MP " << getBrand() << ":" << getModel() << "\n";

}

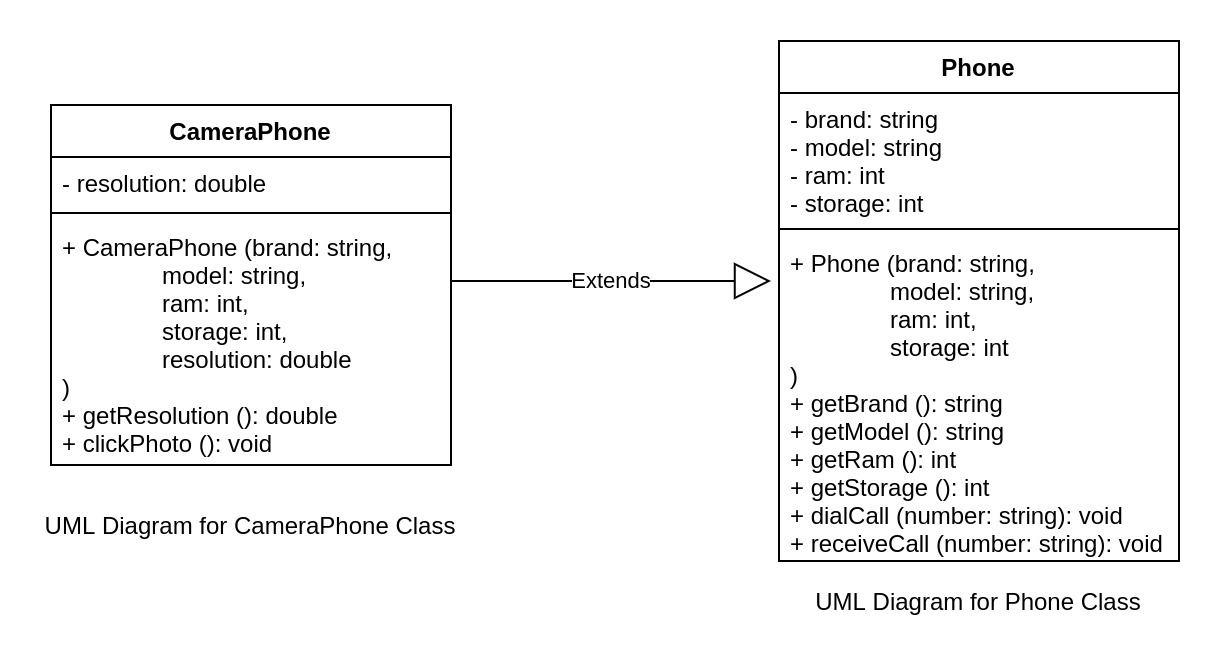
};

Here:

* ***CameraPhone is derived from Phone class.***
* ***CameraPhone "extends" Phone class***.
* *Extends relationship* => **class DerivedClass: public BaseClass**
* **Here, public is *an inheritance-access-specifier***. We will learn more about it later. For now, we will keep using public.
* All the "public" class members of the base class is available in the derived class and its objects.  
  Examples:  
  We can call getBrand() inside clickPhoto().  
  We can call base class methods on the derived class object like this: cameraPhone.getBrand()
* Private class members of the base class are not available in the derived class.
* The constructor of the base class must be called first in the derived class constructor. We need to pass only the required parameters to the base class constructor.  
  It is supposed to be done like this:
* DerivedClass (param1, param2, ...): BaseClass(param1, param2, ….) {
* /\*\* derived class specific initialization \*\*/

}

The UML representation of inheritance is:



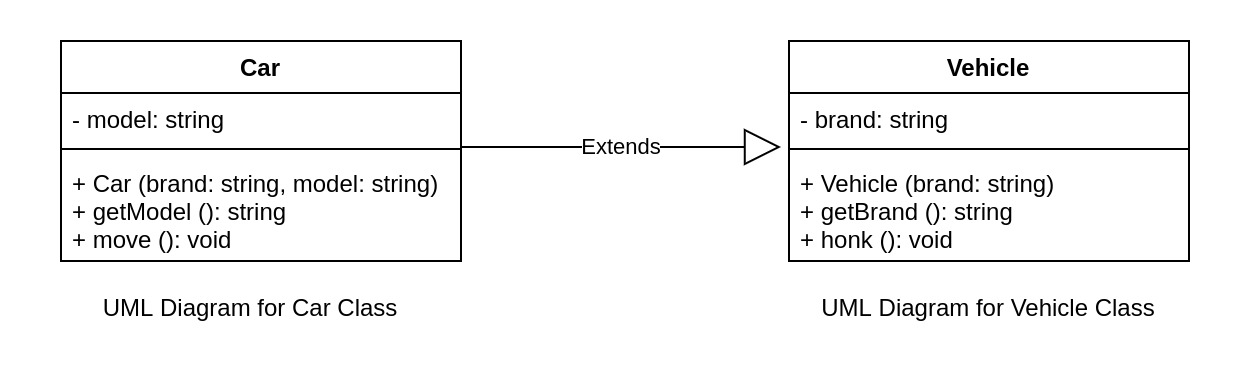
Here, the CameraPhone extends Phone and has class members specific to CameraPhone.

Inheritance UML Representation: An arrow — with a line and a hollow triangle — from the derived class to the base class.

In this section, we learned about Single Inheritance which is a type of Inheritance where one class inherits a single class only.

Let's write some code.

* Create classes based on the below UML diagram.



* Print "{brand} is honking" in honk() method.  
  Example: If brand is Tesla, print "Tesla is honking".
* Print "{brand} {model} is moving" in move() method.  
  Example: If brand is Tesla and model is "Model S", print "Tesla Model S is moving".
* Do not modify the main method.

#### Expected Output

Tesla

Tesla is honking

Tesla Model S

Tesla is honking

Tesla Model S is moving

#include <bits/stdc++.h>

using namespace std;

class Vehicle{

private:

string brand;

public:

Vehicle(){}

Vehicle(string brand){

this->brand = brand;

}

string getBrand(){

return brand;

}

void honk(){

std::cout << brand << " is honking\n";

}

};

class Car: public Vehicle {

private:

string model;

public:

Car(string brand, string model): Vehicle(brand){

this->model = model;

}

string getModel(){

return model;

}

void move(){

std::cout << getBrand() << " " << model << " is moving";

}

};

int main() {

// do not modify the main method

Vehicle vehicle("Tesla");

Car car("Tesla", "Model S");

cout << vehicle.getBrand() << endl;

vehicle.honk();

cout << car.getBrand() << " " << car.getModel() << endl;

car.honk();

car.move();

return 0;

}

# Object-Oriented Programming - Principles in C++: Inheritance

## Hierarchical Inheritance

Which property of OOP is denoted here?

Every Phone is-a ElectronicDevice

Abstraction

Encapsulation

Inheritance✔️

Polymorphism

Correct Answer is Inheritance.

In the following relationship, which of these will be the base class?

Every Phone is-a ElectronicDevice

A. Only Phone

# B. Only ElectronicDevice ✔️

C. Both of these

D. None of these

In the following relationship, which of these will be the derived class?

Every Phone is-a ElectronicDevice

A. Only Phone✔️

B. Only ElectronicDevice

C. Both of these

D. None of these

In Inheritance, X extends Y.

Which of these is true?

A. X: Base Class, Y: Derived Class

B. X: Derived Class, Y: Base Class✔️

C. X: Base Class, Y: Base Class

D. X: Derived Class, Y: Derived Class

Which of these will not be available in the derived class and its object?

A. Public class members of base class

B. Public data properties of base class

C. Private class members of base class✔️

D. Public class methods of base class

Which of these is used to represent inheritance in UML class diagram?

A. An arrow — with a line and a hollow triangle — from the derived class to the base class. ✔️

B. An arrow — with a line and a hollow triangle — from the base class to the derived class.

C. An arrow — with a line and a filled triangle — from the derived class to the base class.

D. An arrow — with a line and a filled triangle — from the base class to the derived class.

In the following relationship, which of these will be the derived class?

Every Phone is-a ElectronicDevice  
Every Camera is-a ElectronicDevice

A. Only Phone

B. Only Camera

C. Only ElectronicDevice

D. Phone and Camera✔️

In the following relationship, whose public members will be available through an object of Phone?

Every Phone is-a ElectronicDevice  
Every Camera is-a ElectronicDevice

A. Only Phone

B. Only ElectronicDevice

C. Both Phone and ElectronicDevice✔️

D. All the three

In the following relationship, through which of these will we be able to access public members of ElectronicDevice?

Every Phone is-a ElectronicDevice  
Every Camera is-a ElectronicDevice

A. Only Phone

B. Only ElectronicDevice

C. Both Phone and Camera

D. All the three✔️

In the following relationship, whose public members will be available through an object of ElectronicDevice?

Every Phone is-a ElectronicDevice  
Every Camera is-a ElectronicDevice

A. Only Phone

B. Only ElectronicDevice✔️

C. Both Phone and ElectronicDevice

D. All the three

In the following relationship, through which of these will we be able to access public members of Phone?

Every Phone is-a ElectronicDevice  
Every Camera is-a ElectronicDevice

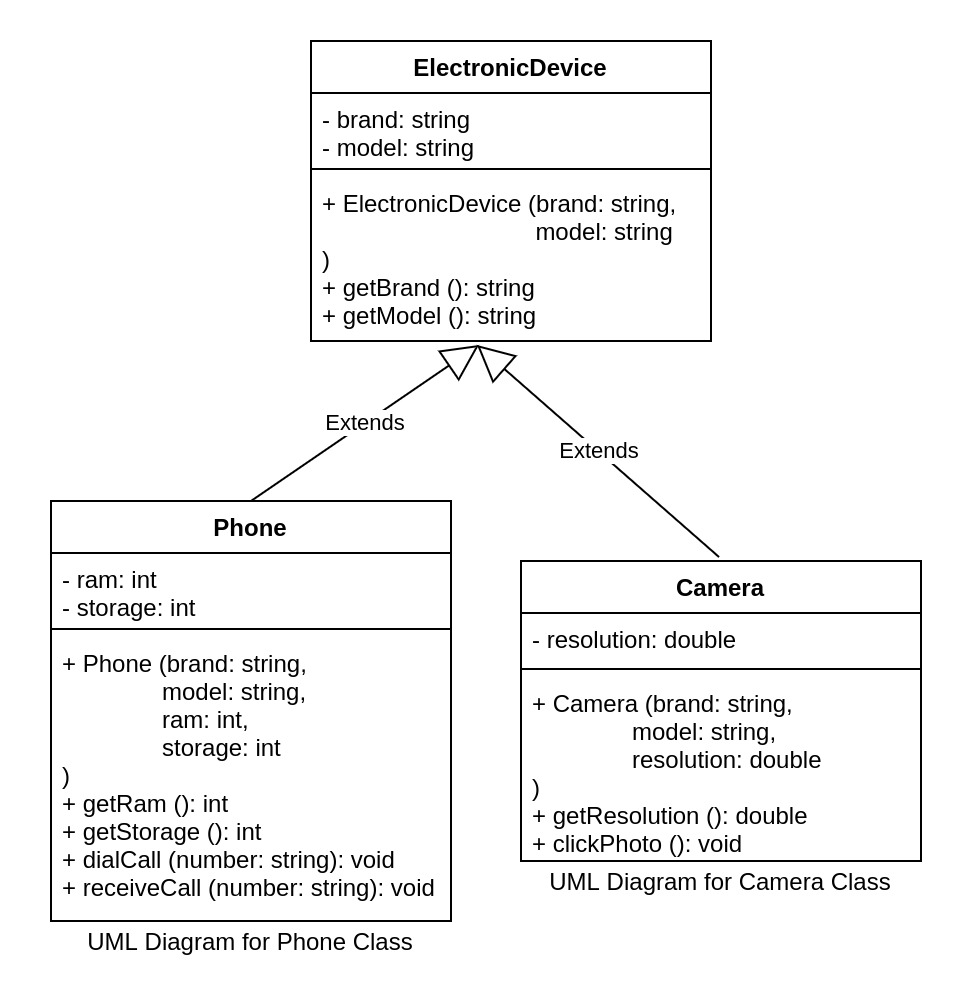
A. Only Phone✔️

B. Only Camera

C. Both Phone and ElectronicDevice

D. All the three

In Single Inheritance, we had one class inheriting from another class. Similarly, we have Hierarchical Inheritance in which multiple classes inherit a single base class.



There can be any number of derived classes and each derived class can have properties different from one another. The code for this will be exactly the same as Single Inheritance.

class ElectronicDevice {

.

.

.

};

class Phone: public ElectronicDevice {

.

.

.

};

class Camera: public ElectronicDevice {

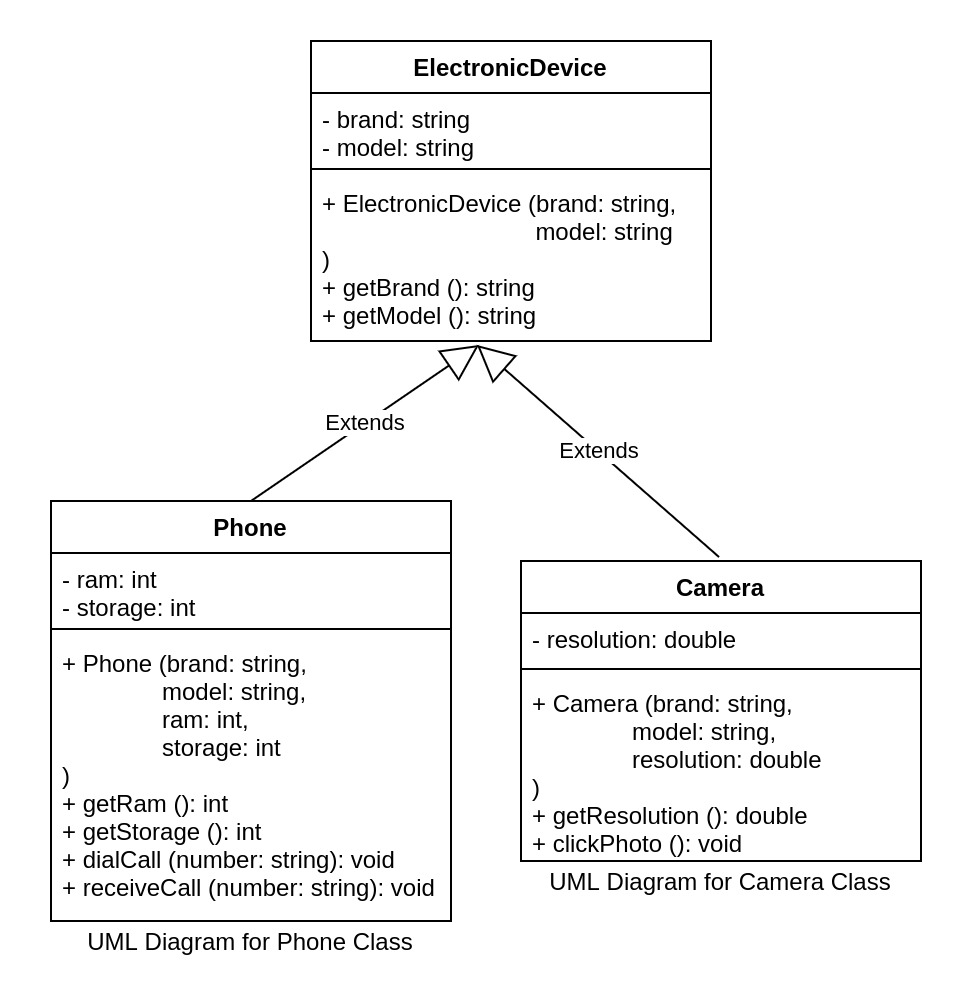
.

.

.

};

Let's write some code.

* Create classes based on the UML diagram.  
  
* Do not modify the main method.

Use the function definitions from [Inheritance:Introduction section](https://workat.tech/courses/object-oriented-programming-principles-c++-yfr9f3liz9c8/inheritance-4kw12t0gusjy/ouhfufclwqxj).

#### Expected Output

Apple iPhone 4 64

Calling 9732130450 from Apple:iPhone

Receiving call from 9732130450 on Apple:iPhone

Canon EOS 1500D 24.1

Clicking photo on a 24.1 MP Canon:EOS 1500D

#include <bits/stdc++.h>

using namespace std;

class ElectronicDevice{

private:

string brand;

string model;

public:

ElectronicDevice(){}

ElectronicDevice(string brand, string model){

this->brand = brand;

this->model = model;

}

string getBrand(){

return brand;

}

string getModel(){

return model;

}

};

class Phone: public ElectronicDevice{

private:

int ram;

int storage;

public:

Phone(string brand, string model, int ram, int storage): ElectronicDevice(brand, model){

this->ram = ram;

this->storage = storage;

}

int getRam(){

return ram;

}

int getStorage(){

return storage;

}

void dialCall(string number){

cout << "Calling " << number << " from " << getBrand() << ":" << getModel() << endl;

}

void receiveCall(string number){

cout << "Receiving call from " << number << " on " << getBrand() << ":" << getModel() << endl;

}

};

class Camera: public ElectronicDevice{

double resolution;

public:

Camera(string brand, string model, double resolution): ElectronicDevice(brand, model){

this->resolution=resolution;

}

double getResolution(){

return resolution;

}

void clickPhoto(){

cout << "Clicking photo on a " << resolution << " MP " << getBrand() << ":" << getModel() << endl << endl;

}

};

int main() {

// do not modify the main method

Phone phone("Apple", "iPhone", 4, 64);

Camera camera("Canon", "EOS 1500D", 24.1);

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

phone.dialCall("9732130450");

phone.receiveCall("9732130450");

cout << camera.getBrand() << " " << camera.getModel() << " " << camera.getResolution() << endl;

camera.clickPhoto();

return 0;

}

# Object-Oriented Programming - Principles in C++: Inheritance

## Multiple Inheritance

1. Which property of OOP is denoted here?

Every Phone is-a ElectronicDevice

Abstraction

Encapsulation

Inheritance✔️

Polymorphism

2. In the following relationship, which of these will be the derived class?

Every Phone is-a ElectronicDevice

A. Only Phone✔

B. Only ElectronicDevice

C. Both of these

D. None of these

Correct Answer is A.

3.In Inheritance, X extends Y.

Which of these is true?

A. X: Base Class, Y: Derived Class

B. X: Derived Class, Y: Base Class ✔

C. X: Base Class, Y: Base Class

D. X: Derived Class, Y: Derived Class

4.Which of these will not be available in the derived class and its object?

A. Public class members of base class

B. Public data properties of base class

C. Private class members of base class ✔

D. Public class methods of base class

5.Which of these is used to represent inheritance in UML class diagram?

A. An arrow — with a line and a hollow triangle — from the derived class to the base class. ✔

B. An arrow — with a line and a hollow triangle — from the base class to the derived class.

C. An arrow — with a line and a filled triangle — from the derived class to the base class.

D. An arrow — with a line and a filled triangle — from the base class to the derived class.

6.In the following relationship, which of these will be the derived class?

Every CameraPhone is-a Phone  
Every CameraPhone is-a Camera

A. Only Phone

B. Only CameraPhone ✔

C. Only Camera

D. Phone and Camera

7. In the following relationship, whose public members will be available through an object of CameraPhone?

Every CameraPhone is-a Phone  
Every CameraPhone is-a Camera

A. Only Phone

B. Only CameraPhone

C. Only Camera

D. All the three ✔

8.In the following relationship, through which of these will we be able to access public members of Camera?

Every CameraPhone is-a Phone  
Every CameraPhone is-a Camera

A. Only Camera

B. Camera and CameraPhone ✔

C. Phone and CameraPhone

D. All the three

9.In the following relationship, whose public members will be available through an object of Camera?

Every CameraPhone is-a Phone  
Every CameraPhone is-a Camera

A. Only Phone

B. Only Camera ✔

C. Only CameraPhone

D. Camera and CameraPhone

10. In the following relationship, through which of these will we be able to access public members of CameraPhone?

Every CameraPhone is-a Phone  
Every CameraPhone is-a Camera

A. Only Phone

B. Only CameraPhone ✔

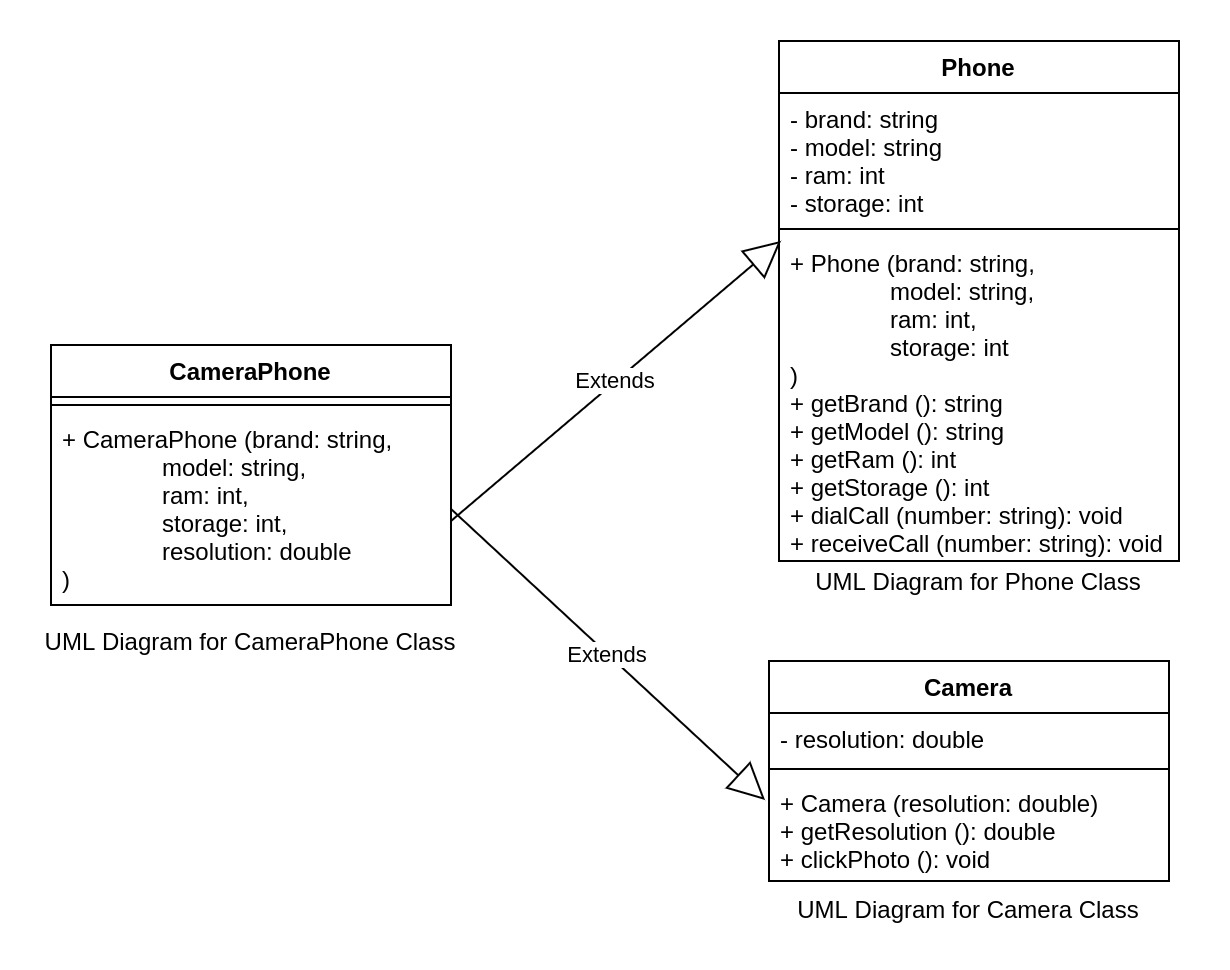
C. Only Camera

D. All the three

In the previous sections, we have seen the following relationship:  
Every CameraPhone is-a Phone

While this is true, it should be noted that the following relationship is also true:  
Every CameraPhone is-a Camera

Therefore, we can say that:  
Every CameraPhone is-a Phone as well is-a Camera

This type of relationship where a class inherits multiple base classes is known as multiple inheritance. It can be represented as:

The code would look something like this:

class Phone {

.

.

.

};

class Camera {

.

.

.

};

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) {

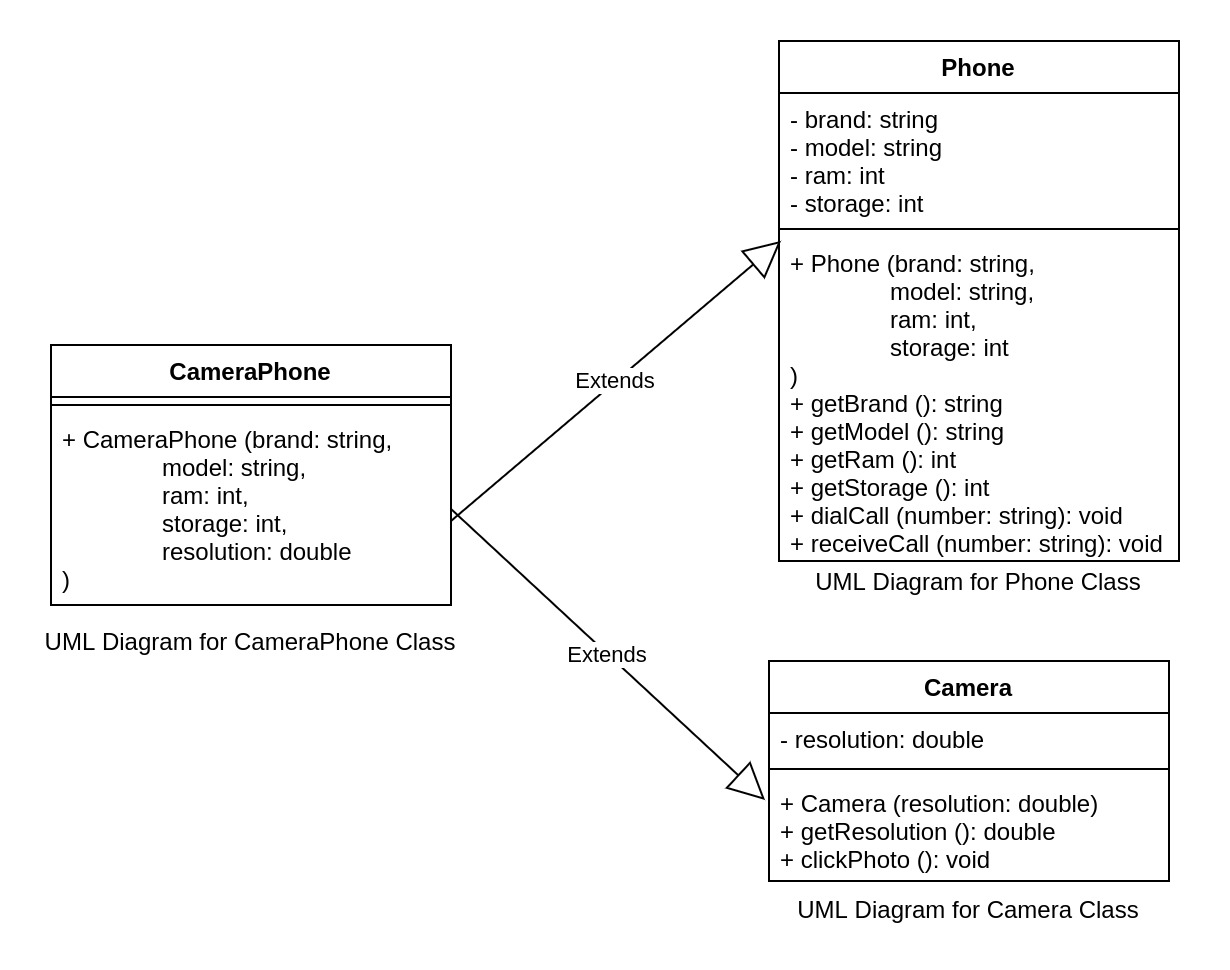
}

};

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.

Let's write the code for the UML diagram.



Use the function definitions from [Inheritance:Introduction section](https://workat.tech/courses/object-oriented-programming-principles-c++-yfr9f3liz9c8/inheritance-4kw12t0gusjy/ouhfufclwqxj).

clickPhoto should print:

Clicking photo on <resolution\_value> MP

##### Example

Clicking photo on 24.1 MP

#### Expected Output

Apple iPhone 4 64

Calling 9732130450 from Apple:iPhone

Receiving call from 9732130450 on Apple:iPhone

24.1

Clicking photo on 24.1 MP

Apple iPhone 8 4 64 12

Calling 9732130450 from Apple:iPhone 8

Receiving call from 9732130450 on Apple:iPhone 8

Clicking photo on 12 MP

#include <bits/stdc++.h>

using namespace std;

class Phone{

private:

string brand;

string model;

int ram;

int storage;

public:

Phone(){}

Phone(string brand, string model, int ram, int storage){

this->brand = brand;

this->model = model;

this->ram = ram;

this->storage = storage;

}

string getBrand() {

return brand;

}

string getModel() {

return model;

}

int getRam() {

return ram;

}

int getStorage() {

return storage;

}

void dialCall (string number) {

cout << "Calling " << number << " from " << brand << ":" << model << "\n";

}

void receiveCall (string number) {

cout << "Receiving call from " << number << " on " << brand << ":" << model << "\n";

}

};

class Camera{

# private:

double resolution;

public:

Camera(){}

Camera(double resolution){

this->resolution = resolution;

}

double getResolution(){

return resolution;

}

void clickPhoto () {

cout << "Clicking photo on a " << resolution << " MP" << endl;

}

};

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)

{

}

};

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;

}

**Output:**

Apple iPhone 4 64

Calling 9732130450 from Apple:iPhone

Receiving call from 9732130450 on Apple:iPhone

24.1

Clicking photo on a 24.1 MP

Apple iPhone 8 4 64 12

Calling 9732130450 from Apple:iPhone 8

Receiving call from 9732130450 on Apple:iPhone 8

Clicking photo on a 12 MP

# Object-Oriented Programming - Principles in C++: Inheritance

## Multilevel Inheritance

In the following relationship, which of these will be the derived class?

Every Phone is-a ElectronicDevice  
Every CameraPhone is-a Phone

A. Only Phone

B. Only CameraPhone

C. Only ElectronicDevice

D. Phone (derived from ElectronicDevice) and CameraPhone (derived from Phone)

Correct Answer is D.

In the following relationship, whose public members will be available through an object of CameraPhone?

Every Phone is-a ElectronicDevice  
Every CameraPhone is-a Phone

A. Only Phone

B. Only CameraPhone

C. Only ElectronicDevice

D. All the three

Correct Answer is D.

In the following relationship, through which of these will we be able to access public members of ElectronicDevice?

Every Phone is-a ElectronicDevice  
Every CameraPhone is-a Phone

A. Only Phone

B. Only CameraPhone

C. Only ElectronicDevice

D. All the three

Correct Answer is D.

In the following relationship, whose public members will be available through an object of ElectronicDevice?

Every Phone is-a ElectronicDevice  
Every CameraPhone is-a Phone

A. Only Phone

B. Only CameraPhone

C. Only ElectronicDevice

D. All the three

Correct Answer is C.

In the following relationship, through which of these will we be able to access public members of CameraPhone?

Every Phone is-a ElectronicDevice  
Every CameraPhone is-a Phone

A. Only Phone

B. Only CameraPhone

C. Only ElectronicDevice

D. All the three

Correct Answer is B.

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

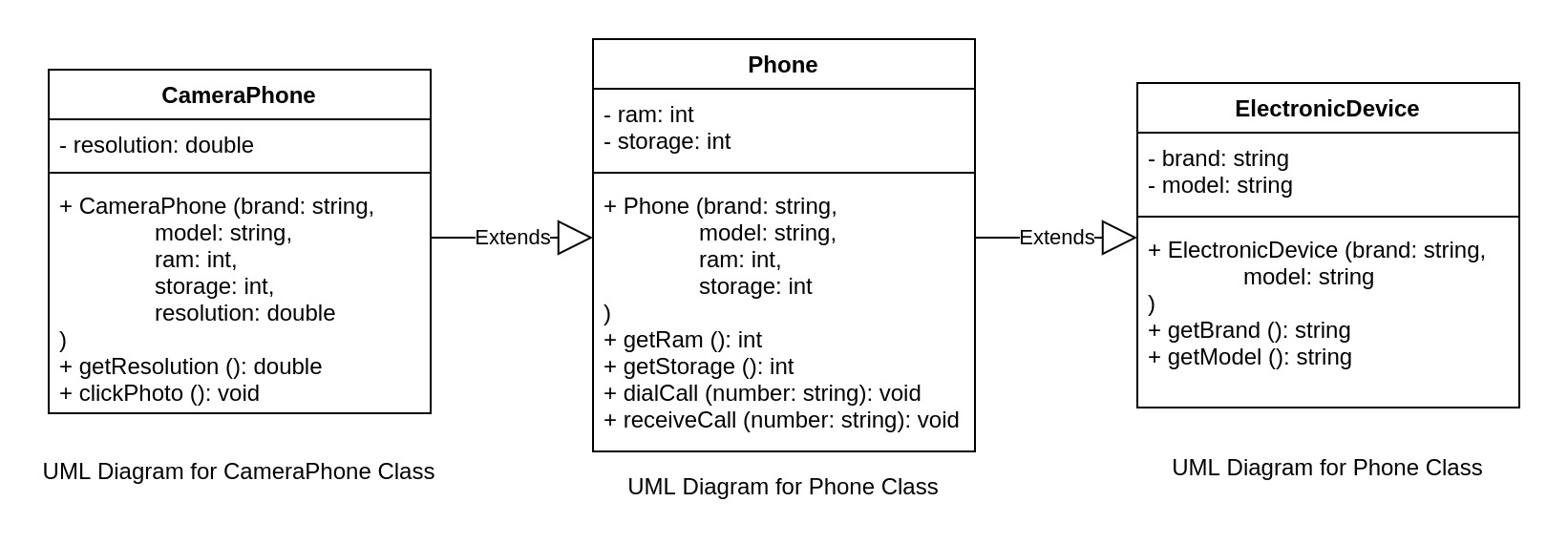
* Every Phone is-a ElectronicDevice
* Every CameraPhone is-a Phone

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:



class ElectronicDevice {

.

.

.

};

class Phone: public ElectronicDevice {

.

.

.

};

class CameraPhone: public Phone {

.

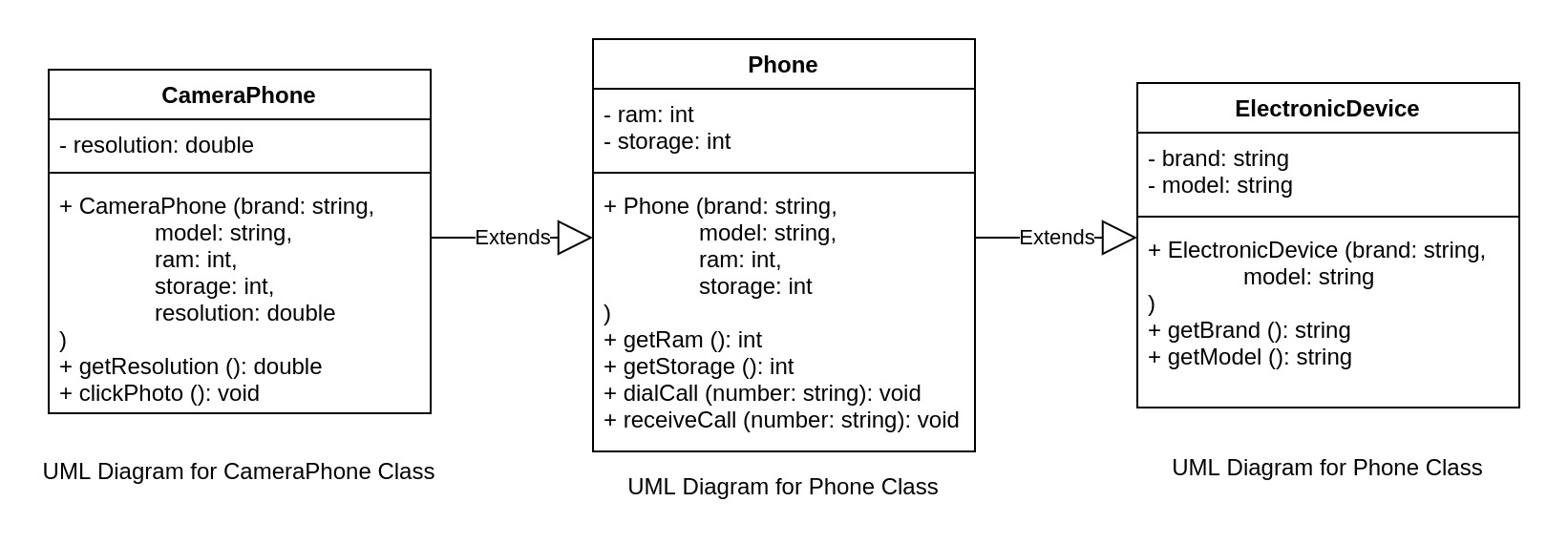
.

.

};

Taking the Car-Vehicle example, SelfDrivingCar can be one class that extends Car.

Let's write some code.

* Create classes based on the UML diagram.  
  
* Do not modify the main method.

Use the function definitions from previous sections.

#### Expected Output

Fitbit Versa 3

Apple iPhone 4 64

Calling 9732130450 from Apple:iPhone

Receiving call from 9732130450 on Apple:iPhone

Apple iPhone 8 4 64 12

Calling 9732130450 from Apple:iPhone 8

Receiving call from 9732130450 on Apple:iPhone 8

Clicking photo on a 12 MP Apple:iPhone 8

# #include <bits/stdc++.h>

# using namespace std;

# class ElectronicDevice {

# private:

# string brand;

# string model;

# public:

# ElectronicDevice(){}

# ElectronicDevice(string brand, string model){

# this->brand = brand;

# this->model = model;

# }

# string getBrand(){

# return brand;

# }

# string getModel(){

# return model;

# }

# };

# class Phone: public ElectronicDevice {

# private:

# int ram;

# int storage;

# public:

# Phone(){}

# Phone(string brand, string model, int ram, int storage): ElectronicDevice(brand, model){

# this->ram = ram;

# this->storage = storage;

# }

# int getRam(){

# return ram;

# }

# int getStorage(){

# return storage;

# }

# void dialCall (string number) {

# cout << "Calling " << number << " from " << getBrand() << ":" << getModel() << "\n";

# }

# 

# void receiveCall (string number) {

# cout << "Receiving call from " << number << " on " << getBrand() << ":" << getModel() << "\n";

# }

# };

# class CameraPhone: public Phone {

# private:

# double resolution;

# public:

# CameraPhone(){}

# CameraPhone(string brand, string model, int ram, int storage, double resolution): Phone(brand, model, ram, storage){

# this->resolution = resolution;

# }

# double getResolution(){

# return resolution;

# }

# void clickPhoto(){

# cout << "Clicking photo on a " << resolution << " MP " << getBrand() << ":" << getModel() << "\n";

# }

# };

# int main() {

# // do not modify the main method

# ElectronicDevice electronicDevice("Fitbit", "Versa 3");

# Phone phone("Apple", "iPhone", 4, 64);

# CameraPhone cameraPhone("Apple", "iPhone 8", 4, 64, 12);

# 

# cout << electronicDevice.getBrand() << " " << electronicDevice.getModel() << endl;

# cout << phone.getBrand() << " " << phone.getModel() << " " << phone.getRam() << " " << phone.getStorage() << endl;

# phone.dialCall("9732130450");

# phone.receiveCall("9732130450");

# 

# cout << cameraPhone.getBrand() << " " << cameraPhone.getModel() << " " << cameraPhone.getRam() << " " << cameraPhone.getStorage() << " " << cameraPhone.getResolution() << endl;

# cameraPhone.dialCall("9732130450");

# cameraPhone.receiveCall("9732130450");

# cameraPhone.clickPhoto();

# return 0;

# }

# Object-Oriented Programming - Principles in C++: Inheritance

## Hybrid Inheritance

In the following relationship, which of these will be the derived class?

Every Phone is-a ElectronicDevice

A. Only Phone

B. Only ElectronicDevice

C. Both of these

D. None of these

Correct Answer is A.

In the following relationship, which of these will be the derived class?

Every Phone is-a ElectronicDevice  
Every Camera is-a ElectronicDevice

A. Only Phone

B. Only Camera

C. Only ElectronicDevice

D. Phone and Camera

Correct Answer is D.

In the following relationship, which of these will be the derived class?

Every CameraPhone is-a Phone  
Every CameraPhone is-a Camera

A. Only Phone

B. Only CameraPhone

C. Only Camera

D. Phone and Camera

Correct Answer is B.

In the following relationship, which of these will be the derived class?

Every Phone is-a ElectronicDevice  
Every CameraPhone is-a Phone

A. Only Phone

B. Only CameraPhone

C. Only ElectronicDevice

D. Phone (derived from ElectronicDevice) and CameraPhone (derived from Phone)

Correct Answer is D.

C

D

Correct Answer is A.

In the following relationship, which of these will be the derived class?

Every Phone is-a ElectronicDevice  
Every Camera is-a ElectronicDevice

A. Only Phone

B. Only Camera

C. Only ElectronicDevice

D. Phone and Camera

A

B

C

D

Correct Answer is D.

In the following relationship, which of these will be the derived class?

Every CameraPhone is-a Phone  
Every CameraPhone is-a Camera

A. Only Phone

B. Only CameraPhone

C. Only Camera

D. Phone and Camera

A

B

C

D

Correct Answer is B.

In the following relationship, which of these will be the derived class?

Every Phone is-a ElectronicDevice  
Every CameraPhone is-a Phone

A. Only Phone

B. Only CameraPhone

C. Only ElectronicDevice

D. Phone (derived from ElectronicDevice) and CameraPhone (derived from Phone)

A

B

C

D

Correct Answer is D.

**Your Score**

4

out of 4

Restart

Apart from single inheritance where we inherit a derived class from a base class, we have learnt that we can:

* Inherit multiple derived classes from a base class
* Inherit a derived class from multiple base classes
* Inherit a derived class from a base class which is derived from some other base class

It seems quite obvious that we can have classes where we have more than one of the above. In fact, we have seen an example of the same previously.

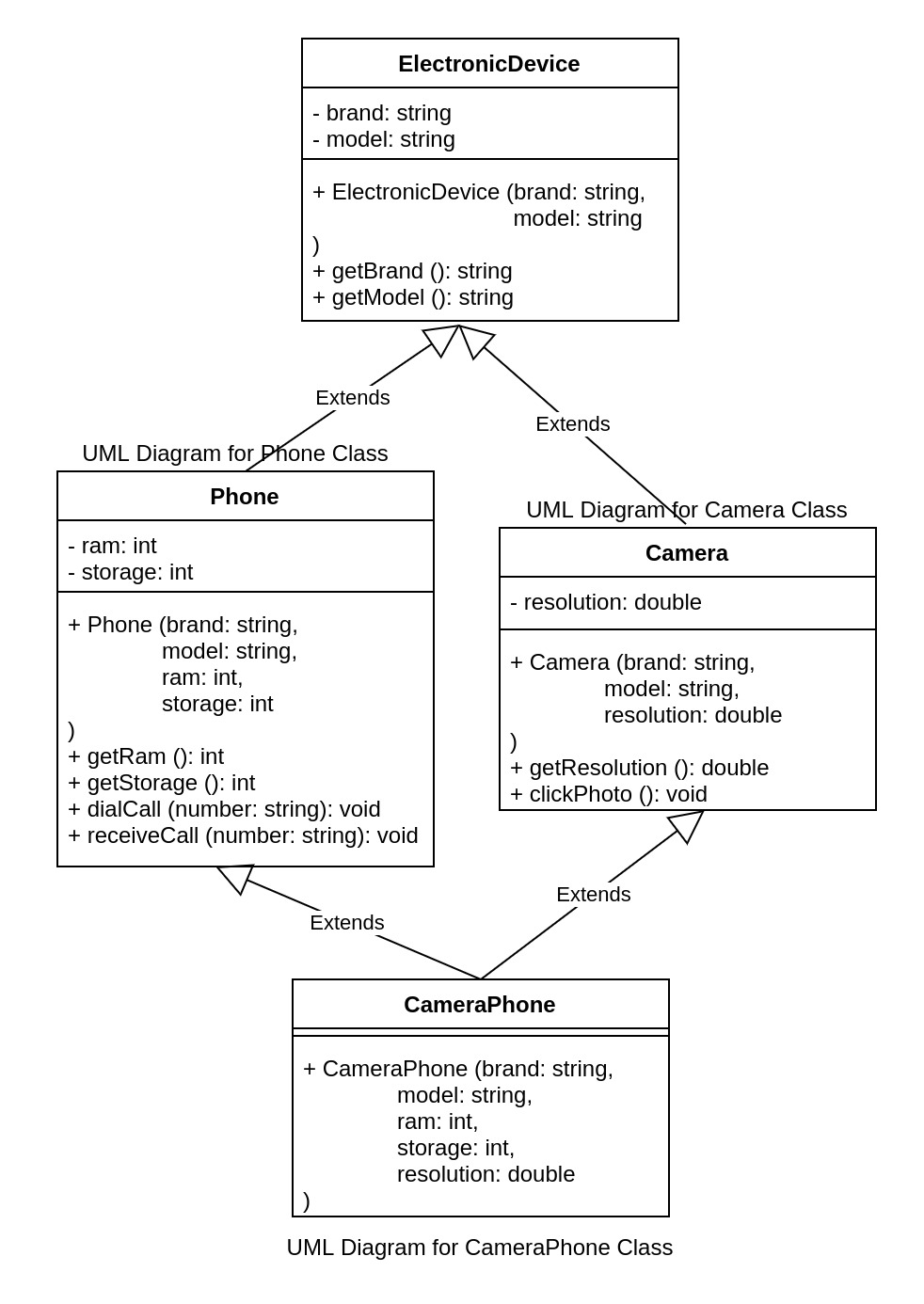
* Every Phone is-a ElectronicDevice
* Every Camera is-a ElectronicDevice
* Every CameraPhone is-a Phone
* Every CameraPhone is-a Camera

Here, we have all of the three types of inheritance discussed above:

* Hierarchical Inheritance: Phone, Camera -> ElectronicDevice
* Multiple Inheritance: CameraPhone -> Phone, Camera
* Multilevel Inheritance: CameraPhone -> Phone -> ElectronicDevice

Such types of inheritance is known as Hybrid Inheritance (Combination of more than 1 type of inheritance).

UML Diagram of the above example:



The code would look something like this:

class ElectronicDevice {

.

.

.

};

class Phone: public ElectronicDevice {

.

.

.

};

class Camera: public ElectronicDevice {

.

.

.

};

class CameraPhone: public Phone, public Camera {

public:

CameraPhone(string brand, string model, int ram, int storage, double resolution): Phone(brand, model, ram, storage), Camera(brand, model, resolution) {

};

# Object-Oriented Programming - Principles in C++: Inheritance

## protected

Which of these will not be available in the derived class and its object?

A. Public class members of base class

B. Public data properties of base class

C. Private class members of base class

D. Public class methods of base class

Correct Answer is C.

As we have learnt previously that a derived class cannot access the private members of the base class. What if we want to use those members in the derived class?

One solution is to make that class member as public instead of private.

But if we do that, we will be exposing that property for any other class to access and compromising on encapsulation.

To resolve this, C++ provides us with another access specifier: protected.

##### Example

class ElectronicDevice {

**protected:**

string brand;

string model;

public:

ElectronicDevice (string brand, string model) {

this->brand = brand;

this->model = model;

}

.

.

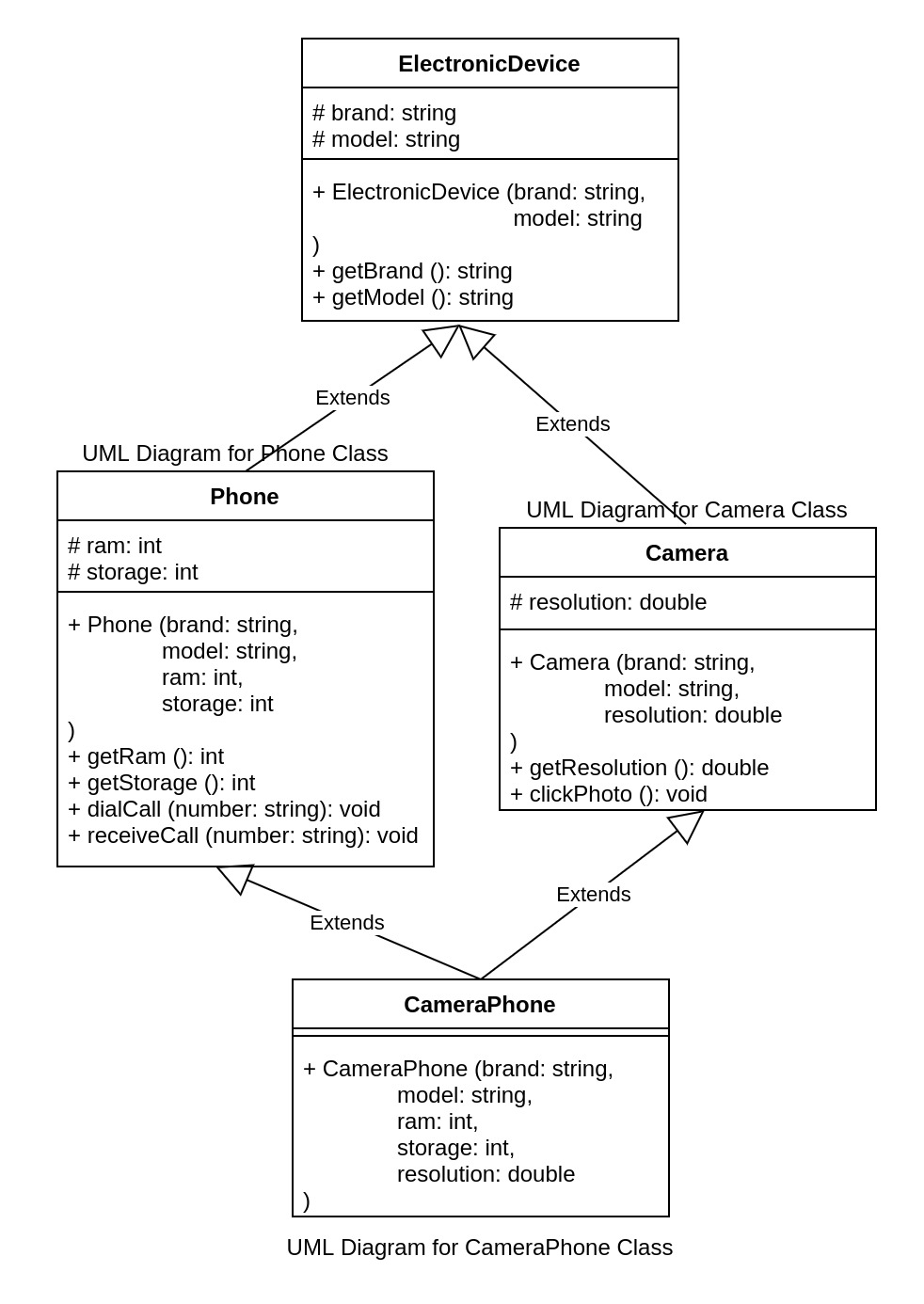
.

.

};

When a class has a particular class member set as protected, the member is accessible to that particular class and any other class which inherits that base class.

Now, in the following relationship, ram and storage will be accessible in CameraPhone but not in the main method. Here, # represents protected.



Let's modify the below code to use protected based on the above UML diagram.

Use the function definitions from previous sections.

#### Expected Output

Fitbit Versa 3

Apple iPhone 4 64

Calling 9732130450 from Apple:iPhone

Receiving call from 9732130450 on Apple:iPhone

Canon EOS 1500D 24.1

Clicking photo on a 24.1 MP Canon:EOS 1500D

Apple iPhone 8 4 64 12

Calling 9732130450 from Apple:iPhone 8

Receiving call from 9732130450 on Apple:iPhone 8

Clicking photo on a 12 MP Apple:iPhone 8

#include <bits/stdc++.h>

using namespace std;

class ElectronicDevice {

protected:

string brand;

string model;

public:

ElectronicDevice (string brand, string model)

{

this->brand = brand;

this->model = model;

}

string getBrand()

{

return brand;

}

string getModel()

{

return model;

}

};

class Phone: public ElectronicDevice

{

protected:

int ram;

int storage;

public:

Phone(string brand,string model,int ram,int storage): ElectronicDevice(brand,model)

{

this->ram=ram;

this->storage=storage;

}

int getRam()

{

return ram;

}

int getStorage()

{

return storage;

}

void dialCall(string number)

{

cout << "Calling " << number << " from " << brand << ":" << model << endl;

}

void receiveCall(string number)

{

cout << "Receiving call from " << number << " on " << brand << ":" << model << endl;

}

};

class Camera: public ElectronicDevice

{

protected:

double resolution;

public:

Camera(string brand,string model,double resolution): ElectronicDevice(brand,model)

{

this->resolution=resolution;

}

double getResolution()

{

return resolution;

}

void clickPhoto()

{

cout<<"Clicking photo on a "<<resolution<<" MP "<<brand<<":"<< model << endl;

}

};

class CameraPhone: public Phone, public Camera

{

public:

CameraPhone(string brand, string model, int ram, int storage, double resolution): Phone(brand, model, ram, storage), Camera(brand, model, resolution) {

}

};

int main() {

// do not modify the main method

ElectronicDevice electronicDevice("Fitbit", "Versa 3");

Phone phone("Apple", "iPhone", 4, 64);

Camera camera("Canon", "EOS 1500D", 24.1);

CameraPhone cameraPhone("Apple", "iPhone 8", 4, 64, 12);

cout << electronicDevice.getBrand() << " " << electronicDevice.getModel() << endl;

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

phone.dialCall("9732130450");

phone.receiveCall("9732130450");

cout << camera.getBrand() << " " << camera.getModel() << " " << camera.getResolution() << endl;

camera.clickPhoto();

cout << cameraPhone.Phone::getBrand() << " " << cameraPhone.Phone::getModel() << " " << cameraPhone.getRam() << " " << cameraPhone.getStorage() << " " << cameraPhone.getResolution() << endl;

cameraPhone.dialCall("9732130450");

cameraPhone.receiveCall("9732130450");

cameraPhone.clickPhoto();

return 0;

}

# Object-Oriented Programming - Principles in C++: Inheritance

## Inheritance Access Specifier

Which of these will not be available in the derived class and its object?

A. Public class members of base class

B. Public data properties of base class

C. Private class members of base class

D. Protected class members of base class

Correct Answer is C.

Which of these will be available in any class through the object of another class A?

A. Public class members of A

B. Protected data properties of A

C. Private class members of A

D. Protected class members of A

Correct Answer is A

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.***

**Inheritance - Quiz 8**

Given:

class BaseClass

{

public:

int public\_prop;

protected:

int protected\_prop;

private:

int private\_prop;

};

class ProtectedDerivedClass: protected BaseClass

{

.

.

.

};

Here, in ProtectedDerivedClass, public\_prop will be:

A. public

B. private

C. protected

D. Not accessible

Correct Answer is C.

class BaseClass

{

public:

int public\_prop;

protected:

int protected\_prop;

private:

int private\_prop;

};

class PublicDerivedClass: public BaseClass

{

/\*\* Here, public\_prop will be public

protected\_prop will be protected

private\_prop will not be accessible \*\*/

};

class ProtectedDerivedClass: protected BaseClass

{

/\*\* Here, public\_prop will be protected

protected\_prop will be protected

private\_prop will not be accessible \*\*/

};

class PrivateDerivedClass: private BaseClass

{

/\*\* Here, public\_prop will be private

protected\_prop will be private

private\_prop will not be accessible \*\*/

};

Given:

class BaseClass

{

public:

int public\_prop;

protected:

int protected\_prop;

private:

int private\_prop;

};

class ProtectedDerivedClass: protected BaseClass

{

.

.

.

};

Here, in ProtectedDerivedClass, protected\_prop will be:

A. public

B. private

C. protected

D. Not accessible

Correct Answer is C.

class BaseClass

{

public:

int public\_prop;

protected:

int protected\_prop;

private:

int private\_prop;

};

class PublicDerivedClass: public BaseClass

{

/\*\* Here, public\_prop will be public

protected\_prop will be protected

private\_prop will not be accessible \*\*/

};

class ProtectedDerivedClass: protected BaseClass

{

/\*\* Here, public\_prop will be protected

protected\_prop will be protected

private\_prop will not be accessible \*\*/

};

class PrivateDerivedClass: private BaseClass

{

/\*\* Here, public\_prop will be private

protected\_prop will be private

private\_prop will not be accessible \*\*/

};

Given:

class BaseClass

{

public:

int public\_prop;

protected:

int protected\_prop;

private:

int private\_prop;

};

class ProtectedDerivedClass: protected BaseClass

{

.

.

.

};

Here, in ProtectedDerivedClass, private\_prop will be:

A. public

B. private

C. protected

D. Not accessible

Correct Answer is D.

class BaseClass

{

public:

int public\_prop;

protected:

int protected\_prop;

private:

int private\_prop;

};

class PublicDerivedClass: public BaseClass

{

/\*\* Here, public\_prop will be public

protected\_prop will be protected

private\_prop will not be accessible \*\*/

};

class ProtectedDerivedClass: protected BaseClass

{

/\*\* Here, public\_prop will be protected

protected\_prop will be protected

private\_prop will not be accessible \*\*/

};

class PrivateDerivedClass: private BaseClass

{

/\*\* Here, public\_prop will be private

protected\_prop will be private

private\_prop will not be accessible \*\*/

};

Given:

class BaseClass

{

public:

int public\_prop;

protected:

int protected\_prop;

private:

int private\_prop;

};

class PrivateDerivedClass: private BaseClass

{

.

.

.

};

Here, in PrivateDerivedClass, public\_prop will be:

A. public

B. private

C. protected

D. Not accessible

Correct Answer is B.

class BaseClass

{

public:

int public\_prop;

protected:

int protected\_prop;

private:

int private\_prop;

};

class PublicDerivedClass: public BaseClass

{

/\*\* Here, public\_prop will be public

protected\_prop will be protected

private\_prop will not be accessible \*\*/

};

class ProtectedDerivedClass: protected BaseClass

{

/\*\* Here, public\_prop will be protected

protected\_prop will be protected

private\_prop will not be accessible \*\*/

};

class PrivateDerivedClass: private BaseClass

{

/\*\* Here, public\_prop will be private

protected\_prop will be private

private\_prop will not be accessible \*\*/

};

Given:

class BaseClass

{

public:

int public\_prop;

protected:

int protected\_prop;

private:

int private\_prop;

};

class PrivateDerivedClass: private BaseClass

{

.

.

.

};

Here, in PrivateDerivedClass, protected\_prop will be:

A. public

B. private

C. protected

D. Not accessible

Correct Answer is B.

class BaseClass

{

public:

int public\_prop;

protected:

int protected\_prop;

private:

int private\_prop;

};

class PublicDerivedClass: public BaseClass

{

/\*\* Here, public\_prop will be public

protected\_prop will be protected

private\_prop will not be accessible \*\*/

};

class ProtectedDerivedClass: protected BaseClass

{

/\*\* Here, public\_prop will be protected

protected\_prop will be protected

private\_prop will not be accessible \*\*/

};

class PrivateDerivedClass: private BaseClass

{

/\*\* Here, public\_prop will be private

protected\_prop will be private

private\_prop will not be accessible \*\*/

};

Given:

class BaseClass

{

public:

int public\_prop;

protected:

int protected\_prop;

private:

int private\_prop;

};

class PrivateDerivedClass: private BaseClass

{

.

.

.

};

Here, in PrivateDerivedClass, private\_prop will be:

A. public

B. private

C. protected

D. Not accessible

Correct Answer is D.

class BaseClass

{

public:

int public\_prop;

protected:

int protected\_prop;

private:

int private\_prop;

};

class PublicDerivedClass: public BaseClass

{

/\*\* Here, public\_prop will be public

protected\_prop will be protected

private\_prop will not be accessible \*\*/

};

class ProtectedDerivedClass: protected BaseClass

{

/\*\* Here, public\_prop will be protected

protected\_prop will be protected

private\_prop will not be accessible \*\*/

};

class PrivateDerivedClass: private BaseClass

{

/\*\* Here, public\_prop will be private

protected\_prop will be private

private\_prop will not be accessible \*\*/

};

# Object-Oriented Programming - Principles in C++: Assessment - I

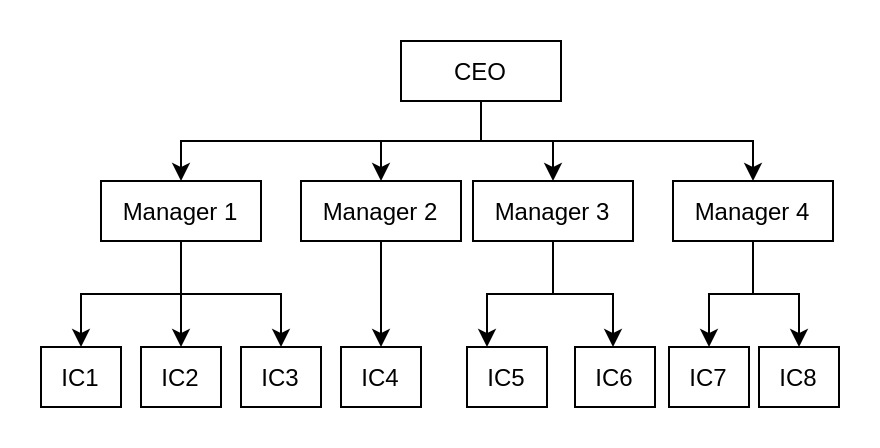
## EMS

Let's suppose that you are building an Employee Management System (EMS) for a company.

There will be employees in the company comprising of individual contributors, managers and a CEO.

* Individual Contributors (ICs) will report to their managers and will not manage anyone.
* Managers will report to their manager (the CEO) and each manager will manage few ICs.
* The CEO will not report to anyone and will manage all the managers.

##### Sample Heirarachy



Here, all ICs, Managers and CEO is an Employee.

Every employee will be assigned an id and will have their name and salary in the EMS apart from additional details. The id and salary will be integers.

The classes that you will be creating are:

* Employee
* IndividualContributor
* Manager
* CEO

Solve the below quiz based on the above information.

##### Important to-do

As you solve the questions, keep writing the class names and their properties on a paper.

After completing the quiz, draw the UML diagram of all the classes and their relationships on paper. Skim through the previous sections if you face any difficulty in drawing the UML diagram.

Please make care to use the correct access specifiers. Mention the getters and setters as well according to the class properties.

Please make sure to do it. It will help you a lot in validating your understanding of the concepts covered till now.

After you are done with the UML diagram, you can verify if it is correct or not by matching it with the UML diagram in the next section. Do not look at the next section before creating the UML diagram yourself.

**Assessment - I - Quiz 1**

What will be the base class here?

A. Employee --ANSWER

B. IndividualContributor

C. Manager

D. CEO

Which of these would be derived from Employee?

A. Only IndividualContributor

B. Only Manager

C. Only CEO

D. All of the above --ANSWER

Which of these data types will you use for id based on the above information?

A. float

B. int --ANSWER

C. double

D. void

Which of these data types will you use for salary based on the above information?

A. float

B. long --ANSWER

C. double

D. void

Which of these data types will you use for name?

A. float

B. int

C. double

D. string --ANSWER

Which of these properties will best represent an IndividualContributor based on the above points?

A. id, name, salary --ANSWER

B. id, name, salary, manager

C. id, name, salary, manager, reportees

D. id, name, salary, reportees

Which of these properties will best represent a Manager based on the above points?

A. id, name, salary

B. id, name, salary, manager

C. id, name, salary, manager, reportees --ANSWER

D. id, name, salary, reportees

Which of these properties will best represent a CEO based on the above points?

A. id, name, salary

B. id, name, salary, manager

C. id, name, salary, manager, reportees

D. id, name, salary, reportees --ANSWER

Which of these should not be a property of the base class Employee?

A. id

B. name

C. salary

D. manager --ANSWER

Which of these should not be a property of the base class Employee?

A. id

B. name

C. salary

D. reportees --ANSWER

Given that the base class Employee will have id, name and salary as the properties, which of these additional properties will you add in the IndividualContributor class?

A. manager --ANSWER

B. reportees

C. None of the above

D. Both manager and reportees

Given that the base class Employee will have id, name and salary as the properties, which of these additional properties will you add in the Manager class?

A. manager

B. reportees

C. None of the above

D. Both manager and reportees --ANSWER

Given that the base class Employee will have id, name and salary as the properties, which of these additional properties will you add in the CEO class?

A. manager

B. reportees --ANSWER

C. None of the above

D. Both manager and reportees

Given that the property 'manager' will store the id of the employee's manager, which of these data types will you use for manager?

A. float

B. int --ANSWER

C. double

D. string

Given that the property 'reportees' will store a list of the id of the employee's reportees.

Which of these data types will you use for reportees?

A. array of ints --ANSWER

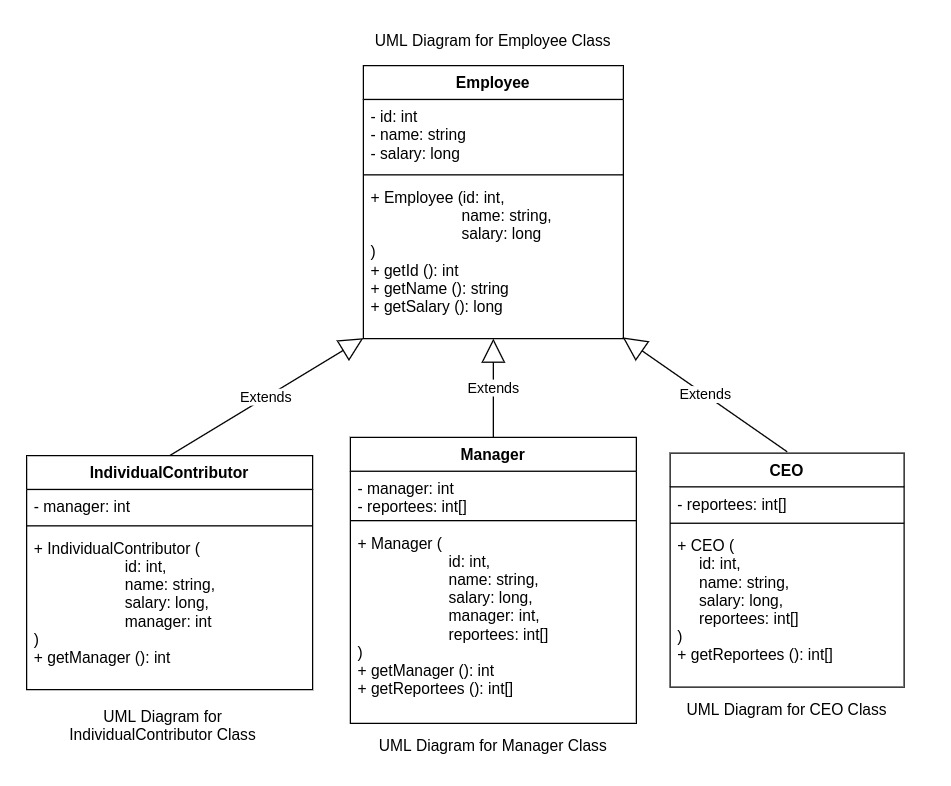
B. int

C. array of strings

D. string

# Object-Oriented Programming - Principles in C++: Assessment - I

## EMS - II



This is the UML Diagram for the Employee Management System that we discussed in the previous section. Let's add more functionalities to it.

We want the system to compute the annual bonus for all the employees. The annual bonus for an employee is 10% of their salary.

##### Example:

Salary: 1234567  
Bonus: 123456.70

Based on the below quiz, add computeBonus method to your UML Diagram.

If we were to add a computeBonus method in any of these classes, which class should we add it to?

A. Employee --ANSWER

B. IndividualContributor

C. Manager

D. CEO

Based on the above example, which of these should be the return type of computeBonus?

A. int

B. float --ANSWER

C. char

D. long

Which of these should be inside the computeBonus method?

A.

return salary\*10;

B.

return salary/10;

C.

return (salary\*10)/100;

D. --ANSWER

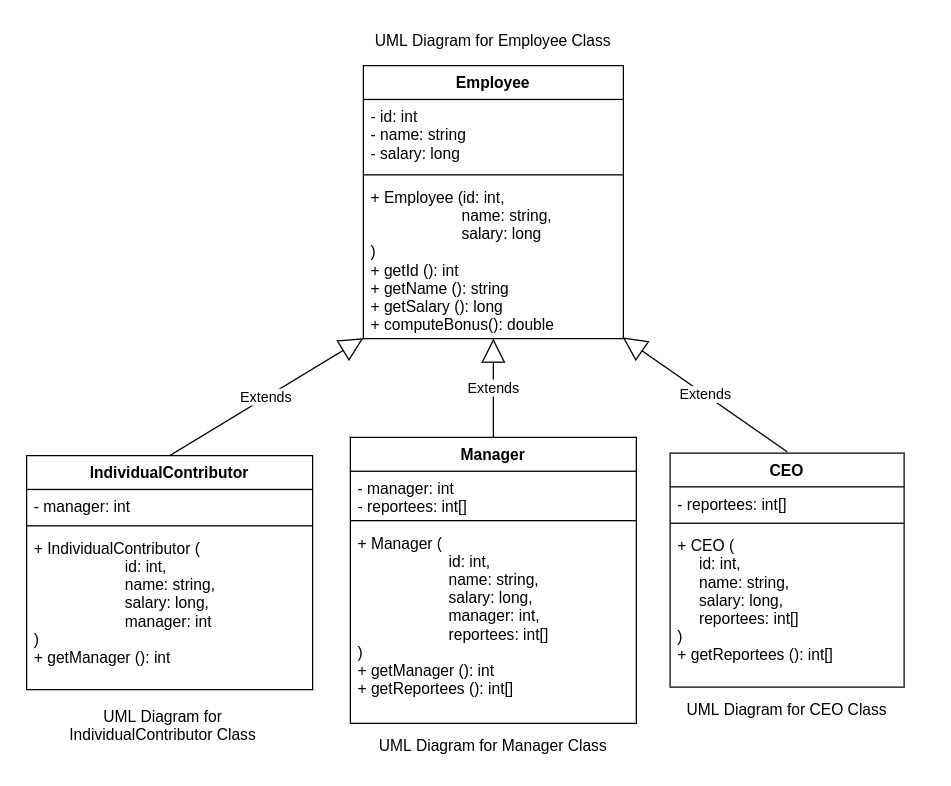
return (salary\*10)/100.0;

# Object-Oriented Programming - Principles in C++: Assessment - I

## Assessment - I

### Problem Statement:

Given UML Diagram for Employee Management System (with Employee, IndividualContributor, Manager and CEO). Create classes for the same with the proper inheritance relationship. Do not modify the main method.



#### Expected Output

1 Phillip Price 10000000

2 Terry Colby 5000000 0

3 Tyrell Wellick 4000000 0

4 Elliot Alderson 100000 2

5 Angela Moss 100000 3

#include <bits/stdc++.h>

using namespace std;

class Employees{

int id;

string name;

long salary;

public:

Employees(){}

Employees(int id, string name, long salary){

this->id = id;

this->name = name;

this->salary = salary;

}

int getId(){

return id;

}

string getName(){

return name;

}

long getSalary(){

return salary;

}

double computeBonus(){

return (salary\*10)/100.0;

}

};

class CEO: public Employees{

int \*ceoReportees;

public:

CEO(){}

CEO(int id, string name, long salary, int \*ceoReportees): Employees(id, name, salary){

ceoReportees = new int[3];

this->ceoReportees=ceoReportees;

}

int getReportees(){

for(int i = 0; i<5; i++){

return ceoReportees[i];

}

}

};

class Manager: public Employees{

private:

int manager;

int \*reportees;

public:

Manager(){}

Manager(int id, string name, long salary, int manager, int \*reportees): Employees(id, name, salary){

reportees = new int[3];

this->manager = manager;

this->reportees = reportees;

}

int getManager(){

return manager;

}

int getReportees(){

for(int i = 0; i<5; i++){

return reportees[i];

}

}

};

class IndividualContributor: public Employees{

private:

int manager;

public:

IndividualContributor(){}

IndividualContributor(int id, string name, long salary, int manager): Employees(id, name, salary){

this->manager = manager;

}

int getManager(){

return manager;

}

};

int main() {

// your code goes here

int ceoReportees[] = {1, 2};

int terryReportees[] = {4};

int tyrellReportees[] = {5};

CEO phillip(1, "Phillip Price", 10000000, ceoReportees);

Manager terry(2, "Terry Colby", 5000000, 0, terryReportees);

Manager tyrell(3, "Tyrell Wellick", 4000000, 0, tyrellReportees);

IndividualContributor elliot(4, "Elliot Alderson", 100000, 2);

IndividualContributor angela(5, "Angela Moss", 100000, 3);

cout << phillip.getId() << " " << phillip.getName() << " " << phillip.getSalary() << endl;

cout << terry.getId() << " " << terry.getName() << " " << terry.getSalary() << " " << terry.getManager() << endl;

cout << tyrell.getId() << " " << tyrell.getName() << " " << tyrell.getSalary() << " " << tyrell.getManager() << endl;

cout << elliot.getId() << " " << elliot.getName() << " " << elliot.getSalary() << " " << elliot.getManager() << endl;

cout << angela.getId() << " " << angela.getName() << " " << angela.getSalary() << " " << angela.getManager() << endl;

return 0;

}

# Object-Oriented Programming - Principles in C++: Polymorphism - I

## Function Overloading

**Polymorphism - I - Quiz 1**

Given two numbers, we can find the maximum of the two numbers like this:

if (firstNumber > secondNumber) {

return firstNumber;

}

return secondNumber;

Let's call this "Logic for finding max of 2 numbers".

How would you write a function to find the maximum of 2 integers?

A.

void max\_num (int firstNumber, int secondNumber) {

//Logic for finding max of 2 numbers

}

B.

int max\_num (int firstNumber, int secondNumber) {

//Logic for finding max of 2 numbers

}

C.

int max\_num (float firstNumber, float secondNumber) {

//Logic for finding max of 2 numbers

}

D.

float max\_num (float firstNumber, float secondNumber) {

//Logic for finding max of 2 numbers

}

Correct Answer is B.

How would you write a function to find the maximum of 2 decimal numbers?

A.

void max\_num (int firstNumber, int secondNumber) {

//Logic for finding max of 2 numbers

}

B.

int max\_num (int firstNumber, int secondNumber) {

//Logic for finding max of 2 numbers

}

C.

int max\_num (float firstNumber, float secondNumber) {

//Logic for finding max of 2 numbers

}

D.

float max\_num (float firstNumber, float secondNumber) {

//Logic for finding max of 2 numbers

}

Correct Answer is D.

Given three numbers, we can find the maximum of the three numbers like this:

if (firstNumber > secondNumber && firstNumber > thirdNumber) {

return firstNumber;

}

if (secondNumber > firstNumber && secondNumber > thirdNumber) {

return secondNumber;

}

return thirdNumber;

Let's call this "Logic for finding max of 3 numbers".

How would you write a function to find the maximum of 3 integers?

A.

void max\_num (int firstNumber, int secondNumber, int thirdNumber) {

//Logic for finding max of 3 numbers

}

B.

int max\_num (int firstNumber, int secondNumber, int thirdNumber) {

//Logic for finding max of 3 numbers

}

C.

int max\_num (float firstNumber, float secondNumber, float thirdNumber) {

//Logic for finding max of 3 numbers

}

D.

float max\_num (float firstNumber, float secondNumber, float thirdNumber) {

//Logic for finding max of 3 numbers

}

Correct Answer is B.

How would you write a function to find the maximum of 3 decimal numbers?

A.

void max\_num (int firstNumber, int secondNumber, int thirdNumber) {

//Logic for finding max of 3 numbers

}

B.

int max\_num (int firstNumber, int secondNumber, int thirdNumber) {

//Logic for finding max of 3 numbers

}

C.

int max\_num (float firstNumber, float secondNumber, float thirdNumber) {

//Logic for finding max of 3 numbers

}

D.

float max\_num (float firstNumber, float secondNumber, float thirdNumber) {

//Logic for finding max of 3 numbers

}

Correct Answer is D.

Let's say that we want to create a program where:

* We can find the maximum of two integers
* We can find the maximum of two decimals (double)
* We can find the maximum of three integers
* We can find the maximum of three decimals (double)

Since it is not possible to create functions with the same variable names in C, what we can do here is:

int max\_two\_integers (int firstNumber, int secondNumber) {

//Logic for finding max of 2 numbers

}

double max\_two\_decimals (double firstNumber, double secondNumber) {

//Logic for finding max of 2 numbers

}

int max\_three\_integers (int firstNumber, int secondNumber, int thirdNumber) {

//Logic for finding max of 3 numbers

}

double max\_three\_decimals (double firstNumber, double secondNumber, double thirdNumber) {

//Logic for finding max of 3 numbers

}

Does not look good, right?

C++ was created to allow writing clean, structured code and so it allows us to create functions with the same name (with different parameters).

In C++, we can do something like this:

int max\_num (int firstNumber, int secondNumber) {

//Logic for finding max of 2 numbers

}

double max\_num (double firstNumber, double secondNumber) {

//Logic for finding max of 2 numbers

}

int max\_num (int firstNumber, int secondNumber, int thirdNumber) {

//Logic for finding max of 3 numbers

}

double max\_num (double firstNumber, double secondNumber, double thirdNumber) {

//Logic for finding max of 3 numbers

}

Let's write a code with 4 functions named max\_num each with the following logic:

* Find the maximum of two integers
* Find the maximum of two decimals (double)
* Find the maximum of three integers
* Find the maximum of three decimals (double)

In the main method call these with the following values and print the return values.

4 12

1.618 2.71828

4 54 12

3.14159 1.618 2.71828

#### Expected Output

12

2.71828

54

3.14159

#include <bits/stdc++.h>

using namespace std;

int max\_num (int firstNumber, int secondNumber) {

//Logic for finding max of 2 numbers

if (firstNumber>secondNumber)

return firstNumber;

else

return secondNumber;

}

double max\_num (double firstNumber, double secondNumber) {

//Logic for finding max of 2 numbers

if (firstNumber>secondNumber)

return firstNumber;

else

return secondNumber;

}

int max\_num (int firstNumber, int secondNumber, int thirdNumber) {

//Logic for finding max of 3 numbers

if (firstNumber>secondNumber && firstNumber>thirdNumber)

return firstNumber;

else if (secondNumber>firstNumber && secondNumber>thirdNumber)

return secondNumber;

else

return thirdNumber;

}

double max\_num (double firstNumber, double secondNumber, double thirdNumber) {

//Logic for finding max of 3 numbers

if (firstNumber>secondNumber && firstNumber>thirdNumber)

return firstNumber;

else if (secondNumber>firstNumber && secondNumber>thirdNumber)

return secondNumber;

else

return thirdNumber;

}

int main() {

// your code goes here

cout<< max\_num(4,12) << "\n";

cout<< max\_num(1.618,2.71828)<< "\n";

cout<< max\_num(4,54,12)<< "\n";

cout<< max\_num(3.14159,1.618,2.71828)<< "\n";

return 0;

}

This is known as ***function overloading.***

***Function overloading is the ability to create multiple functions with the same name but different implementations.***

What basically happens here is that when the code is getting compiled, the compiler tries to select the correct method for each of function calls based on the arguments being passed (number of arguments and data types of those arguments).

Note that this is not valid:

int divide (int numerator, int denominator) {

return numerator/denominator;

}

float divide (int numerator, int denominator) {

return numerator/((float) denominator);

}

Here, even though the implementation and return type is different, the number of parameters and the data types are still the same. That's why it is not valid. ***Function Overloading allows us to create functions with the same name with different parameters only.***

**Polymorphism - I - Quiz 2**

How do you create an object of class Phone with a constructor which takes brand, model, ram, and storage as parameters?

A.

Phone phone ("OnePlus", "8", 16, 64);

B.

Phone phone = ("OnePlus", "8", 16, 64);

C.

Phone = Phone ("OnePlus", "8", 16, 64);

D.

Phone = phone ("OnePlus", "8", 16, 64);

Correct Answer is A.

Go through the below code properly.

Phone getBetterPhone (Phone firstPhone, Phone secondPhone) {

if (firstPhone.getRam() > secondPhone.getRam()) {

return firstPhone;

} else if (firstPhone.getRam() < secondPhone.getRam()) {

return secondPhone;

} else if (firstPhone.getStorage() > secondPhone.getStorage()) {

return firstPhone;

}

return secondPhone;

}

What of the following is false in regard to this method?

A. Always returns secondPhone

B. Prioritizes better ram over storage

C. Compares storage only if ram is same

D. If ram is same, better storage wins

Correct Answer is A.

Let's write another code. This time with 3 functions named 'sum' each with the following logic:

* Find the sum of two integers
* Find the sum of two decimals
* Find the sum of two complex numbers.

Since there is no data type for complex numbers, I have already created a class for complex number so that you don't have to.

A complex number is a number which has a real part and an imaginary part represented as:

a + bi

Here, a and b are numbers. a represents the real part and bi represents the imaginary part.  
i is the imaginary unit (a suffix used in the imaginary part).

##### Example

3 + 5i

Here, 3 is the real part and 5i is the imaginary part.

You can read more about complex numbers [here](https://en.wikipedia.org/wiki/Complex_number) if you want to. Though the information mentioned in this section is sufficient for this course.

Sum of two complex numbers is another complex number with the real part as the sum of the real part of the two objects and the imaginary part should be the sum of the imaginary part of the two objects.

(a + bi) + (c + di) => (a + c) + (b + d)i

##### Example

(1 + 3i) + (4 + 8i) => 5 + 11i

In the main method, the variables has been created. Call the methods on them and print the return values.

* 7 11
* 3.14 2.718
* Complex Number objects (1 + 3i) and (4 + 8i)

#### Expected Output

18

5.858

5 + 11i

Use the getFormatted method to get the complex number in this format.

#include <bits/stdc++.h>

using namespace std;

class ComplexNumber {

private:

int real;

int imaginary;

public:

ComplexNumber (int real, int imaginary) {

this->real = real;

this->imaginary = imaginary;

}

int getRealPart () {

return real;

}

int getImaginaryPart () {

return imaginary;

}

//You can use this method on a ComplexNumber object to get the string value

//Use case: cout << complexNumber.getFormatted();

string getFormatted() {

//to\_string is an in-built method to convert a number to string

//This will return 3 + 7i for real:3 and imaginary:7

return to\_string(real) + " + " + to\_string(imaginary) + "i";

}

};

// Find the sum of two integers

// Find the sum of two decimals

// Find the sum of two complex numbers.

int sum1(int firstInteger, int secondInteger){

return firstInteger+secondInteger;

}

double sum1(double firstDouble, double secondDouble){

return firstDouble+secondDouble;

}

string sum2(int real\_part, int imaginary\_part){

return to\_string(real\_part) + " + " + to\_string(imaginary\_part) + "i";

}

int main() {

// your code goes here

int firstInteger = 7, secondInteger = 11;

double firstDouble = 3.14, secondDouble = 2.718;

int firstReal = 1, firstImaginary = 3;

int secondReal = 4, secondImaginary = 8;

//Create two ComplexNumber objects using (firstReal and firstImaginary) and (secondReal and secondImaginary)

ComplexNumber C1(firstReal, firstImaginary);

ComplexNumber C2(secondReal, secondImaginary);

//Call and Print the sum methods for firstInteger and secondInteger

cout << sum1(firstInteger, secondInteger) << "\n";

//Call and Print the sum methods for firstDouble and secondDouble

cout << sum1(firstDouble, secondDouble) << "\n";

int real\_part = C1.getRealPart()+C2.getRealPart();

int imaginary\_part = C1.getImaginaryPart()+C2.getImaginaryPart();

//Call and Print the sum methods for firstComplex and secondComplex

cout << sum2(real\_part, imaginary\_part) << "\n";

return 0;

}

# Object-Oriented Programming - Principles in C++: Polymorphism - I

## Operator Overloading - I

Given two complex numbers, find its sum.

Sum of two complex numbers is another complex number with the real part as the sum of the real part of the two objects and the imaginary part should be the sum of the imaginary part of the two objects.

3 + 4i

1 + 11i

A. 14 + 5i

B. 4 + 15i

C. 5 + 14i

D. 15 + 4i

orrect Answer is B.

How do you find the sum of two integers in C?

int firstNumber = 5;

int secondNumber = 11;

int sum = 0;

A.

sum = firstNumber + secondNumber;

B.

sum = firstNumber - secondNumber;

C.

sum = firstNumber \* secondNumber;

D.

sum = firstNumber / secondNumber;

Correct Answer is A.

How do you find the sum of two decimal numbers in C?

float firstNumber = 1.618;

float secondNumber = 3.14;

float sum = 0;

A.

sum = firstNumber + secondNumber;

B.

sum = firstNumber - secondNumber;

C.

sum = firstNumber \* secondNumber;

D.

sum = firstNumber / secondNumber;

rrect Answer is A.

We've learned about complex numbers and how to find its sum in the previous section. When we can use + to get the sum of two integers, decimals, etc, why can't we do the same with complex numbers?

Because it is not an in-built data type.

If we do firstComplexNumber + secondComplexNumber, the compiler won't understand how to do it because it just knows that these are objects of class ComplexNumber and don't know anything about how to do any operation on them.

So what if we can tell the compiler how to do an operation on it?

C++ allows us to do that through something known as "operator overloading".

Operator overloading is similar to function overloading in a way that instead of the same function name, we have the same operator behaving differently depending on the parameters.

We can assume that the internal implementation of + operator is somethign like this:

//Here "operator +" denotes a function which does + operation on the parameters

int operator + (int firstNumber, int secondNumber) {

//Some internal implementation

}

float operator + (float firstNumber, float secondNumber) {

//Some internal implementation

}

long operator + (long firstNumber, long secondNumber) {

//Some internal implementation

}

double operator + (double firstNumber, double secondNumber) {

//Some internal implementation

}

Now let's create an overloaded + operator for class ComplexNumber as well.

This is the sum method that we created in the previous section:

ComplexNumber sum (ComplexNumber firstNumber, ComplexNumber secondNumber) {

int real\_part = firstNumber.getRealPart() + secondNumber.getRealPart();

int imaginary\_part = firstNumber.getImaginaryPart() + secondNumber.getImaginaryPart();

return ComplexNumber(real\_part, imaginary\_part);

}

We can modify the same to do an operator overloading.

Replace the function name from "sum" with "operator +" in the below code and hit run.

Note that you need to mention both operator and +.

operator +

#### Expected Output

3+9i

#include <bits/stdc++.h>

using namespace std;

class ComplexNumber {

private:

int real;

int imaginary;

public:

ComplexNumber (int real, int imaginary) {

this->real = real;

this->imaginary = imaginary;

}

int getRealPart () {

return real;

}

int getImaginaryPart () {

return imaginary;

}

string getFormatted() {

//to\_string is an in-built method to convert a number to string

//This will return 3+7i for real:3 and imaginary:7

return to\_string(real) + "+" + to\_string(imaginary) + "i";

}

};

ComplexNumber operator + (ComplexNumber firstNumber, ComplexNumber secondNumber) {

int real\_part = firstNumber.getRealPart() + secondNumber.getRealPart();

int imaginary\_part = firstNumber.getImaginaryPart() + secondNumber.getImaginaryPart();

return ComplexNumber(real\_part, imaginary\_part);

}

int main() {

// your code goes here

ComplexNumber complexNumber(1, 3), complexNumber2(2, 6);

cout << (complexNumber + complexNumber2).getFormatted();

return 0;

}

What basically happens here is that when the code is getting compiled, the compiler tries to select the correct implementation for each of the operations based on the arguments (operands) being passed (operated on). There is an implementation present internally for each of the operations that we've been doing until now on in-built data types. So during compilation, the compiler tries to find the correct implementation. If it is unable to find for the set of operands then it gives an error.

Note that operator overloading is not limited to arithmetic operators. **Almost** all operators including arithmetic, relational, logical, assignment, etc can be overloaded.

Syntax of operator overloading is:

return\_type operator op (data\_type param1, ...) {

}

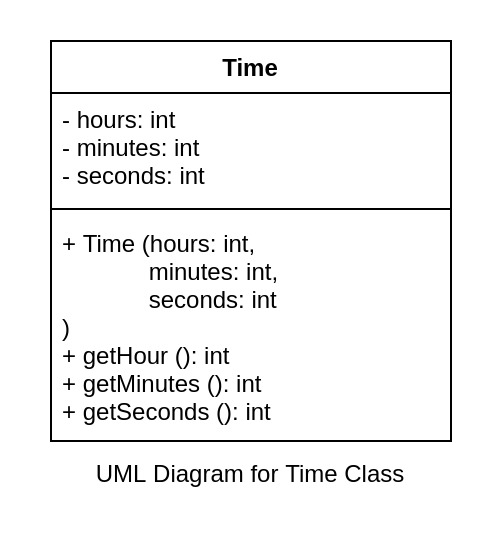
In the above example:

ComplexNumber operator + (ComplexNumber firstNumber, ComplexNumber secondNumber) {

//Addition Logic

}

This type of function is known as operator function. The function name is "operator" followed by the operator symbol.

* Let's write a code.
* Given UML diagram for Time class:  
  
* Overload the greater than operator (>) for two Time Objects.
* Given two time objects timeFirst and timeSecond, timeFirst > timeSecond should return 1 if:
  + timeFirst's hours is greater than timeSecond's hours
  + timeFirst's hours is equal to timeSecond's but timeFirst's minutes is greater
  + timeFirst's hours and minutes are equal to timeSecond's but timeFirst's seconds is greater
* Note that you can use int as the return type for > operator and return 1 if first parameter is greater than second parameter otherwise return 0.
* Do not modify the main method.

#### Expected Output

1 0

#include <bits/stdc++.h>

using namespace std;

class Time{

private:

int hours;

int minutes;

int seconds;

public:

Time(){}

Time(int hours, int minutes, int seconds){

this->hours = hours;

this->minutes = minutes;

this->seconds = seconds;

}

int getHours(){

return hours;

}

int getMinutes(){

return minutes;

}

int getSeconds(){

return seconds;

}

};

int operator > (Time t1, Time t2) {

if(t1.getHours()>t2.getHours())

return 1;

else if(t1.getHours()==t2.getHours() && t1.getMinutes()>t2.getMinutes())

return 1;

else if(t1.getHours()==t2.getHours() && t1.getMinutes()==t2.getMinutes() && t1.getSeconds() > t2.getSeconds())

return 1;

else

return 0;

}

int main() {

// your code goes here

Time t1(15, 10, 35);

Time t2(12, 45, 55);

cout << (t1 > t2) << " " << (t2 > t1);

return 0;

}

# Object-Oriented Programming - Principles in C++: Polymorphism - I

## Compile-time Polymorphism

Which of these is not allowed in C++ based on what has been covered till now?

A. Same func. name, different no. of params

B. Same func. name, no. of params but different params data types

C. Same func. name, no. of params, params data types but different return types

D. Different func. names

Correct Answer is C.

What is the name of the feature in C++ that allows functions to exist in different forms (with different implementations) in the same class/file?

A. Function Overloading

B. Operator Overloading

C. Function Overriding

D. Operator Overriding

Correct Answer is A.

What is the name of the feature in C++ that allows operators to exist in different forms (with different implementations) in the same class/file?

A. Function Overloading

B. Operator Overloading

C. Function Overriding

D. Operator Overriding

orrect Answer is B.

The word Polymorphism means having many forms (Poly: Many, Morph: Form).

Have we seen any form of polymorphism till now in this course?

Function Overloading???

Operator Overloading???

Yes, both function overloading and operator overloading are forms of polymorphism.

* Function overloading allows us to have a function in different forms (with different implementations)
* Operator overloading allows us to have an operator in different forms (with different implementations)

In both function overloading and operator overloading, the compiler decides which implementation of the function/operator to choose for a particular function call or operation.

This type of polymorphism is known as compile-time polymorphism.

Why is it not just called polymorphism?

It has a specific name 'Compile-time Polymorphism' because there is another type of polymorphism where the implementation is not decided during compile time. We'll learn about it in a later chapter.

# Object-Oriented Programming - Principles in C++: Polymorphism - I

## Function Hiding

What is the name of the feature in C++ that allows functions to exist in different forms (with different implementations) in the same class/file?

A. Function Overloading

B. Operator Overloading

C. Function Overriding

D. Operator Overriding

Correct Answer is A.

Which of the following is true for the below code?

class MobilePhone {

public:

void print() {

cout << "MobilePhone" << endl;

}

};

class Laptop {

public:

void print() {

cout << "Laptop" << endl;

}

};

A. This is not valid as both classes have print function with same number of parameters

B. This is valid because the functions are in different independent classes

C. Calling print on an object of Laptop will print MobilePhone

D. Calling print on an object of MobilePhone will print Laptop

Correct Answer is B.

Which of these will not be available in the derived class and its object?

A. Public class members of base class

B. Public data properties of base class

C. Private class members of base class

D. Public class methods of base class

orrect Answer is C.

As we learned earlier that we can have functions with the same name but differerent parameters through function overloading.

We already know that we can have functions with the same method name and same parameters in different classes if the classes are independent of each other. Go through the below code and run it to see what happens.

#include <bits/stdc++.h>

using namespace std;

class MobilePhone {

public:

void print() {

cout << "MobilePhone" << endl;

}

};

class Laptop {

public:

void print() {

cout << "Laptop" << endl;

}

};

int main() {

MobilePhone mobilePhone;

Laptop laptop;

mobilePhone.print(); //This will print "MobilePhone"

laptop.print(); //This will print "Laptop"

return 0;

}

**Output:**

MobilePhone

Laptop

Since these classes are independent of each other, one class does not care about which methods are there in the other class.

Now, let's say there are 2 classes and both of them are related through inheritance. Go through the below code and run it to see what happens.

#include <bits/stdc++.h>

using namespace std;

class BaseClass {

public:

void print() {

cout << "Base Class" << endl;

}

};

class DerivedClass: BaseClass {

public:

void print() {

cout << "Derived Class" << endl;

}

};

int main() {

BaseClass base;

DerivedClass derived;

base.print(); //This will print "Base Class"

derived.print(); //This will print "Derived Class"

return 0;

}

**Output:**

Base Class

Derived Class

Here, we are seeing the same behavior as the case when there was no relationship. The function of the respective class is called. But there is a different reason for it.

Go through the below code properly and read on. Do not run it yet.

#include <bits/stdc++.h>

using namespace std;

class BaseClass {

public:

void print() {

cout << "Base Class" << endl;

}

void print(int num) {

cout << "Base Class " << num << endl;

}

void print(string name) {

cout << "Base Class " << name << endl;

}

};

class DerivedClass: BaseClass {

public:

void print() {

cout << "Derived Class" << endl;

}

};

int main() {

BaseClass base;

DerivedClass derived;

base.print(); //This will print "Base Class"

base.print(42); //This will print "Base Class 42"

base.print("workat.tech"); //This will print "Base Class workat.tech"

derived.print(); //This will print "Derived Class"

derived.print(42); //This should ideally print "Base Class 42"

derived.print("workat.tech"); //This should ideally print "Base Class workat.tech"

return 0;

}

**Compilation Error:**

code.cpp: In function 'int main()':

code.cpp:34:18: error: no matching function for call to 'DerivedClass::print(int)'

34 | derived.print(42); //This should ideally print "Base Class 42"

| ^

code.cpp:21:7: note: candidate: 'void DerivedClass::print()'

21 | void print() {

| ^~~~~

code.cpp:21:7: note: candidate expects 0 arguments, 1 provided

code.cpp:35:29: error: no matching function for call to 'DerivedClass::print(const char [12])'

35 | derived.print("workat.tech"); //This should ideally print "Base Class workat.tech"

| ^

code.cpp:21:7: note: candidate: 'void DerivedClass::print()'

21 | void print() {

| ^~~~~

code.cpp:21:7: note: candidate expects 0 arguments, 1 provided

Here we have created a function named print:

* in the derived class
* with the same name as multiple functions in the base class

Ideally, all the public members (properties and functions) of the base class should be available through the derived class as well (because of inheritance).

So the expected behavior here is:

* base.print(); //This will print "Base Class"
* base.print(42); //This will print "Base Class 42"
* base.print("workat.tech"); //This will print "Base Class workat.tech"
* derived.print(); //This will print "Derived Class"
* derived.print(42); //This should ideally print "Base Class 42"
* derived.print("workat.tech"); //This should ideally print "Base Class workat.tech"

Now run the above code.

After running it, we can see that we get a compilation error because:

* any function named print from the base class
* won't be available through the derived class

This is not the expected behavior, right?

This is known as function hiding. Whenever a function with the same name is present in both base class and derived class, the functions with that name in the base class won't be available through the derived class irrespective of the parameters.

This is the reason why "Base Class" was getting printed on doing base.print() and "Derived Class" on doing derived.print() in this code:

BaseClass base;

DerivedClass derived;

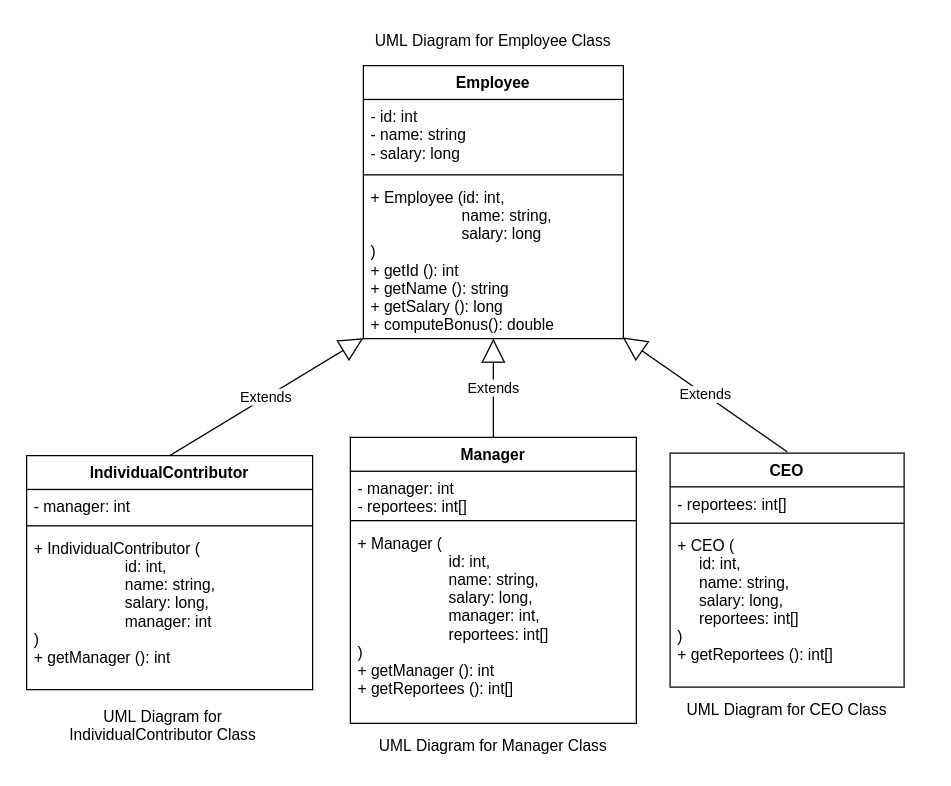
base.print(); //This will print "Base Class"

derived.print(); //This will print "Derived Class"

**Object-Oriented Programming - Principles in C++: Polymorphism - I**

**EMS - III**

Let's get back to our previous example of Employee Management System.



What if we want to have different bonus computation for ICs, Managers and CEO?

New Bonus Structure:

* IC: 10% of salary
* Manager: if salary > 30,00,000 => (1,50,000 + 20% of salary) else 25% of salary
* CEO: 500000 + 50% of salary
* **Polymorphism - I - Quiz 6**
* If we want the above feature in our system, in which class(es) should we have the computeBonus method? Choose the one which makes most sense.
* A. Only Employee
* B. Manager, CEO
* C. Employee, IndividualContributor, Manager
* D. Employee, IndividualContributor, Manager, CEO

Correct Answer is D.

What would be the logic inside the computeBonus method in IndividualContributor class?

A.

return (salary\*25)/100.0;

B.

return 150000 + (salary\*20)/100.0;

C.

return 500000 + (salary\*50)/100.0;

D.

return (salary\*10)/100.0;

orrect Answer is D.

What would be the logic inside the computeBonus method in CEO class?

A.

return (salary\*25)/100.0;

B.

return 150000 + (salary\*20)/100.0;

C.

return 500000 + (salary\*50)/100.0;

D.

return (salary\*10)/100.0;

Correct Answer is C.

What would be the logic inside the computeBonus method in Manager class?

A.

return (salary\*25)/100.0;

B.

return 150000 + (salary\*20)/100.0;

C.

if (salary > 3000000) {

return 150000 + (salary\*20)/100.0;

} else {

return (salary\*25)/100.0;

}

D.

if (salary > 3000000) {

return (salary\*25)/100.0;

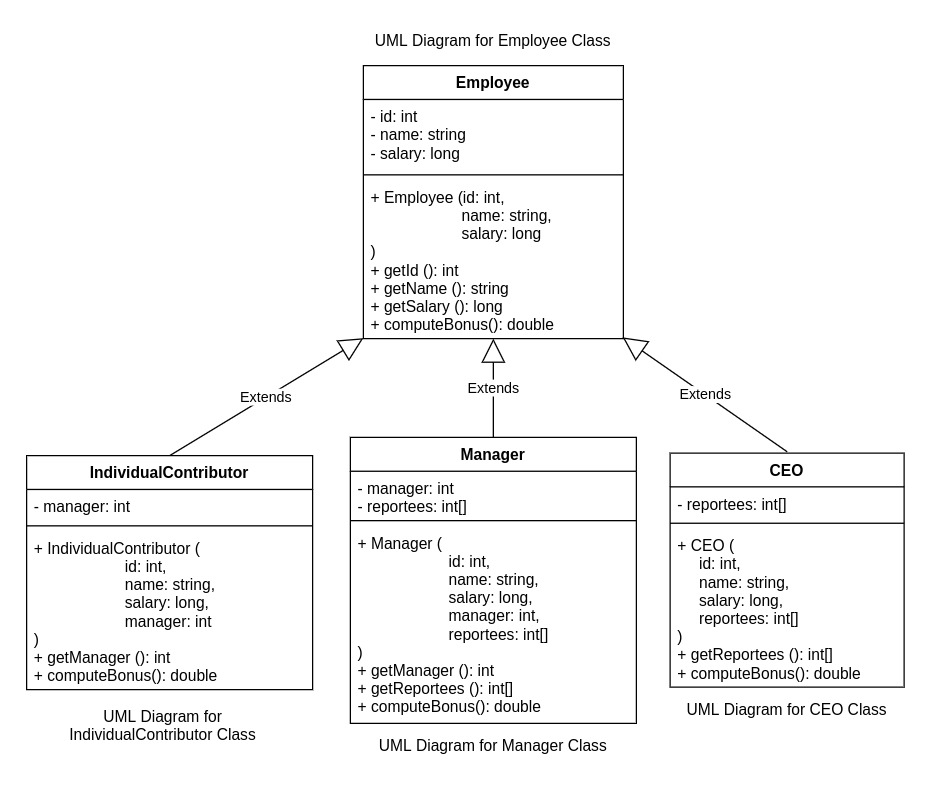
} else {

return 150000 + (salary\*20)/100.0;

}

Correct Answer is C.

New UML Diagram:

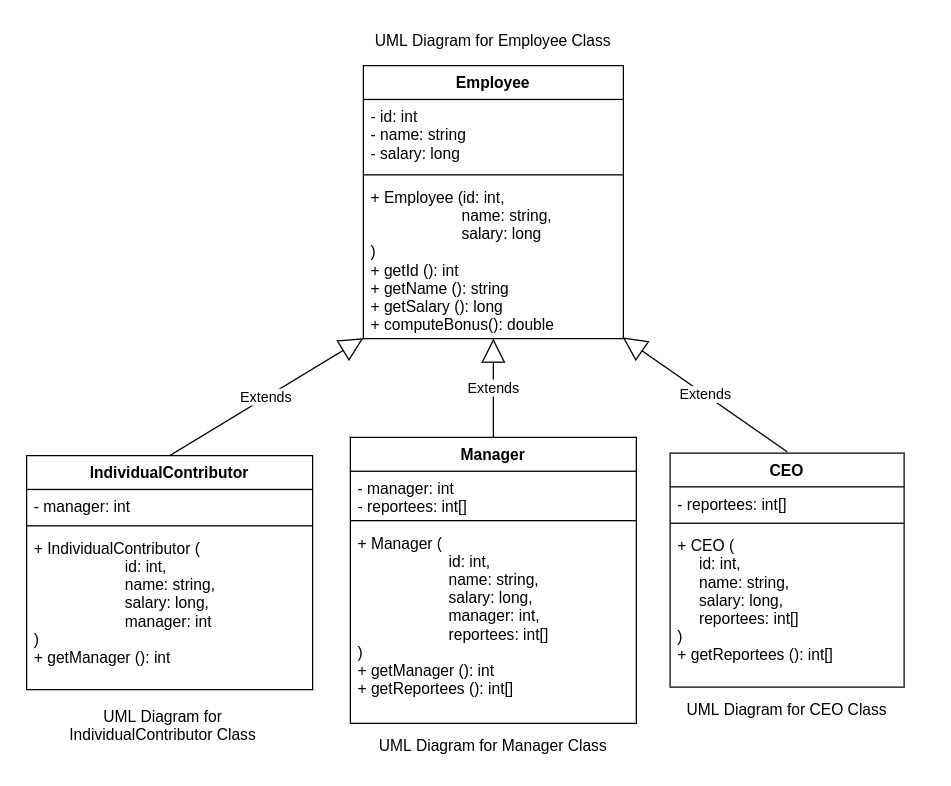


# Object-Oriented Programming - Principles in C++: Assessment - II, III & IV

## Assessment - II

### Problem Statement:

Given IndividualContributor, Manager and CEO classes (Code prefilled in the editor)



Create three functions named max each with the following definition:

* Takes two IndividualContributor objects as parameters and returns the object with the maximum salary.
* Takes two Manager objects as parameters and returns the object with the maximum salary.
* Takes two CEO objects as parameters and returns the object with the maximum salary.

Do not modify the main method.

#### Expected Output

20000000 20000000

5000000 5000000

100000 100000

#include <bits/stdc++.h>

using namespace std;

class Employee {

int id;

string name;

long salary;

public:

Employee(int id, string name, long salary) {

this->id = id;

this->name = name;

this->salary = salary;

}

int getId() {

return id;

}

string getName() {

return name;

}

long getSalary() {

return salary;

}

double computeBonus() {

return (salary\*100)/10.0;

}

};

class IndividualContributor: public Employee {

int manager;

public:

IndividualContributor(int id, string name, long salary, int manager): Employee(id, name, salary) {

this->manager = manager;

}

int getManager() {

return manager;

}

};

class Manager: public Employee {

int manager;

int \*reportees;

public:

Manager(int id, string name, long salary, int manager, int \*reportees): Employee(id, name, salary) {

this->manager = manager;

this->reportees = reportees;

}

int getManager() {

return manager;

}

int\* getReportees() {

return reportees;

}

};

class CEO: public Employee {

int \*reportees;

public:

CEO(int id, string name, long salary, int \*reportees): Employee(id, name, salary) {

this->reportees = reportees;

}

int\* getReportees() {

return reportees;

}

};

IndividualContributor max(IndividualContributor IC1, IndividualContributor IC2){

if(IC1.getSalary()>IC2.getSalary())

return IC1;

else

return IC2;

}

Manager max(Manager M1, Manager M2){

if(M1.getSalary()>M2.getSalary())

return M1;

else

return M2;

}

CEO max(CEO ceo1, CEO ceo2){

if(ceo1.getSalary()>ceo2.getSalary())

return ceo1;

else

return ceo2;

}

int main() {

// your code goes here

int ceoReportees[] = {1, 2};

int terryReportees[] = {4};

int tyrellReportees[] = {5};

CEO lester(0, "Lester Moore", 20000000, ceoReportees);

CEO phillip(1, "Phillip Price", 10000000, ceoReportees);

Manager terry(2, "Terry Colby", 5000000, 0, terryReportees);

Manager tyrell(3, "Tyrell Wellick", 4000000, 0, tyrellReportees);

IndividualContributor elliot(4, "Elliot Alderson", 100000, 2);

IndividualContributor angela(5, "Angela Moss", 100000, 3);

cout << max(phillip, lester).getSalary() << " " << max(lester, phillip).getSalary() << endl;

cout << max(terry, tyrell).getSalary() << " " << max(tyrell, terry).getSalary() << endl;

cout << max(elliot, angela).getSalary() << " " << max(angela, elliot).getSalary() << endl;

return 0;

}

**Output:**

20000000 20000000

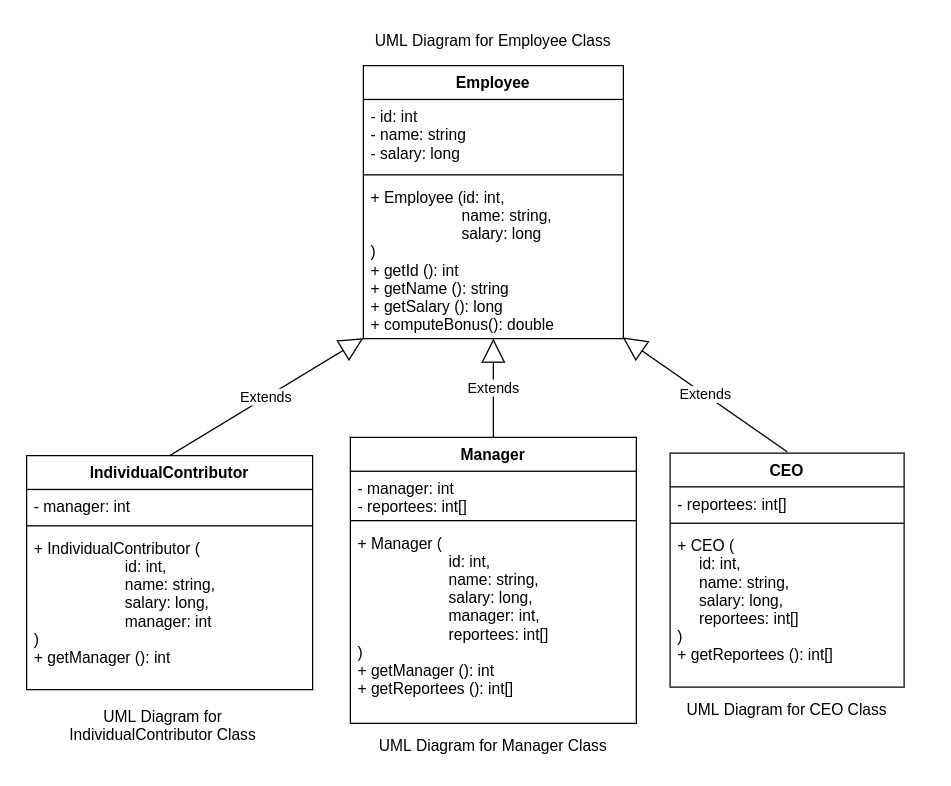
5000000 5000000

100000 100000

## Assessment - III

### Problem Statement:

Given IndividualContributor, Manager and CEO class. Code prefilled in the editor.



Overload the greater than operator (>) each with the following definition:

* Takes two IndividualContributor objects as parameters and returns 1 if first object's salary is greater otherwise returns 0.
* Takes two Manager objects as parameters and returns 1 if first object's salary is greater otherwise returns 0.
* Takes two CEO objects as parameters and returns 1 if first object's salary is greater otherwise returns 0.

Do not modify the main method.

#### Expected Output

0 1

1 0

0 0

#include <bits/stdc++.h>

using namespace std;

class Employee {

int id;

string name;

long salary;

public:

Employee(int id, string name, long salary) {

this->id = id;

this->name = name;

this->salary = salary;

}

int getId() {

return id;

}

string getName() {

return name;

}

long getSalary() {

return salary;

}

double computeBonus() {

return (salary\*100)/10.0;

}

};

class IndividualContributor: public Employee {

int manager;

public:

IndividualContributor(int id, string name, long salary, int manager): Employee(id, name, salary) {

this->manager = manager;

}

int getManager() {

return manager;

}

};

class Manager: public Employee {

int manager;

int \*reportees;

public:

Manager(int id, string name, long salary, int manager, int \*reportees): Employee(id, name, salary) {

this->manager = manager;

this->reportees = reportees;

}

int getManager() {

return manager;

}

int\* getReportees() {

return reportees;

}

};

class CEO: public Employee {

int \*reportees;

public:

CEO(int id, string name, long salary, int \*reportees): Employee(id, name, salary) {

this->reportees = reportees;

}

int\* getReportees() {

return reportees;

}

};

int operator > (IndividualContributor IC1, IndividualContributor IC2) {

if(IC1.getSalary()>IC2.getSalary())

return 1;

else

return 0;

}

int operator > (Manager M1, Manager M2){

if(M1.getSalary()>M2.getSalary())

return 1;

else

return 0;

}

int operator > (CEO ceo1, CEO ceo2){

if(ceo1.getSalary()>ceo2.getSalary())

return 1;

else

return 0;

}

int main() {

// your code goes here

int ceoReportees[] = {1, 2};

int terryReportees[] = {4};

int tyrellReportees[] = {5};

CEO lester(0, "Lester Moore", 20000000, ceoReportees);

CEO phillip(1, "Phillip Price", 10000000, ceoReportees);

Manager terry(2, "Terry Colby", 5000000, 0, terryReportees);

Manager tyrell(3, "Tyrell Wellick", 4000000, 0, tyrellReportees);

IndividualContributor elliot(4, "Elliot Alderson", 100000, 2);

IndividualContributor angela(5, "Angela Moss", 100000, 3);

cout << (phillip > lester) << " " << (lester > phillip) << endl;

cout << (terry > tyrell) << " " << (tyrell > terry) << endl;

cout << (elliot > angela) << " " << (angela > elliot) << endl;

return 0;

}

**Output:**

0 1

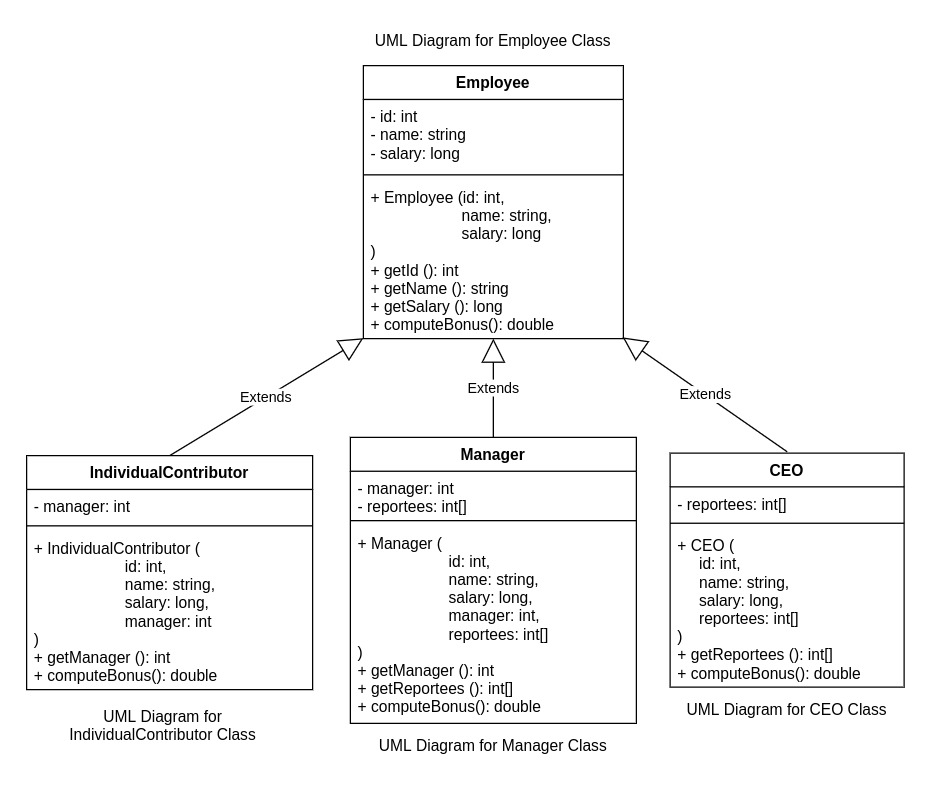
1 0

0 0

## Assessment - IV

### Problem Statement:

Update this code to use the following logic.



New Bonus Structure:

* IC: 10% of salary
* Manager: if salary > 30,00,000 => (1,50,000 + 20% of salary) else 25% of salary
* CEO: 500000 + 50% of salary

#### Expected Output

10500000.00 5500000.00

1150000.00 950000.00

10000.00 10000.00

#include <bits/stdc++.h>

using namespace std;

class Employee {

int id;

string name;

long salary;

public:

Employee(int id, string name, long salary) {

this->id = id;

this->name = name;

this->salary = salary;

}

int getId() {

return id;

}

string getName() {

return name;

}

long getSalary() {

return salary;

}

double computeBonus() {

return (salary\*10)/100.0;

}

};

class IndividualContributor: public Employee {

int manager;

public:

IndividualContributor(int id, string name, long salary, int manager): Employee(id, name, salary) {

this->manager = manager;

}

int getManager() {

return manager;

}

double computeBonus() {

return (getSalary()\*10)/100.0;

}

};

class Manager: public Employee {

int manager;

int \*reportees;

public:

Manager(int id, string name, long salary, int manager, int \*reportees): Employee(id, name, salary) {

this->manager = manager;

this->reportees = reportees;

}

int getManager() {

return manager;

}

int\* getReportees() {

return reportees;

}

double computeBonus() {

if (getSalary() > 3000000) {

return 150000 + (getSalary()\*20)/100.0;

} else {

return (getSalary()\*25)/100.0;

}

}

};

class CEO: public Employee {

int \*reportees;

public:

CEO(int id, string name, long salary, int \*reportees): Employee(id, name, salary) {

this->reportees = reportees;

}

int\* getReportees() {

return reportees;

}

double computeBonus() {

int bonus = 500000+((getSalary()\*50)/100.0);

return bonus;

}

};

int main() {

// your code goes here

int ceoReportees[] = {1, 2};

int terryReportees[] = {4};

int tyrellReportees[] = {5};

CEO lester(0, "Lester Moore", 20000000, ceoReportees);

CEO phillip(1, "Phillip Price", 10000000, ceoReportees);

Manager terry(2, "Terry Colby", 5000000, 0, terryReportees);

Manager tyrell(3, "Tyrell Wellick", 4000000, 0, tyrellReportees);

IndividualContributor elliot(4, "Elliot Alderson", 100000, 2);

IndividualContributor angela(5, "Angela Moss", 100000, 3);

cout << fixed << setprecision(2);

cout << lester.computeBonus() << " " << phillip.computeBonus() << endl;

cout << terry.computeBonus() << " " << tyrell.computeBonus() << endl;

cout << elliot.computeBonus() << " " << angela.computeBonus() << endl;

return 0;

}

**Output:**

10500000.00 5500000.00

1150000.00 950000.00

10000.00 10000.00

# Object-Oriented Programming - Principles in C++: Polymorphism - II

## Function Templates

What is the name of the feature in C++ that allows functions to exist in different forms (with different implementations) in the same class/file?

A. Function Overloading

B. Operator Overloading

C. Function Overriding

D. Operator Overriding

Correct Answer is A.

We have previously learned that we can have multiple functions with the same name but different parameters in the same file/class through Function Overloading.

##### Example

int max\_num (int firstNumber, int secondNumber) {

if (firstNumber > secondNumber) {

return firstNumber;

}

return secondNumber;

}

long max\_num (long firstNumber, long secondNumber) {

if (firstNumber > secondNumber) {

return firstNumber;

}

return secondNumber;

}

float max\_num (float firstNumber, float secondNumber) {

if (firstNumber > secondNumber) {

return firstNumber;

}

return secondNumber;

}

double max\_num (double firstNumber, double secondNumber) {

if (firstNumber > secondNumber) {

return firstNumber;

}

return secondNumber;

}

int main() {

int firstInt = 5, secondInt = 7;

float firstFloat = 3.14159, secondFloat = 2.71828;

long firstLong = 54362478632846, secondLong = 7148342783278;

double firstDouble = 2.71828182845, secondDouble = 3.1415926535;

cout << max\_num(firstInt, secondInt) << endl;

cout << max\_num(firstFloat, secondFloat) << endl;

cout << max\_num(firstLong, secondLong) << endl;

cout << max\_num(firstDouble, secondDouble) << endl;

}

This is a very valid use case but the issue here is that we have to write a lot of code with the exact same logic. If you notice carefully, the only difference between these functions is the data types being used. Each of the above functions can be logically written as:

data\_type max\_num (data\_type firstNumber, data\_type secondNumber) {

if (firstNumber > secondNumber) {

return firstNumber;

}

return secondNumber;

}

Won't it be great if we can just write like this and the compiler takes care of all the other stuff. Turns out that C++ actually allows us to achieve this type of polymorphism through something known as TEMPLATES!

Let's see it in action. Run the below code:

#include <bits/stdc++.h>

using namespace std;

template <class data\_type>

data\_type max\_num (data\_type firstNumber, data\_type secondNumber) {

if (firstNumber > secondNumber) {

return firstNumber;

}

return secondNumber;

}

int main() {

int firstInt = 5, secondInt = 7;

float firstFloat = 3.14159, secondFloat = 2.71828;

long firstLong = 54362478632846, secondLong = 7148342783278;

double firstDouble = 2.71828182845, secondDouble = 3.1415926535;

cout << max\_num(firstInt, secondInt) << endl;

cout << max\_num(firstFloat, secondFloat) << endl;

cout << max\_num(firstLong, secondLong) << endl;

cout << max\_num(firstDouble, secondDouble) << endl;

return 0;

}

**Output:**

7

3.14159

54362478632846

3.14159

As you can see that we have been able to replace the four functions with just a single function by using templates. Let's look at how to use templates:

template <class identifier>

Here,

* template is a keyword to tell the compiler that we are creating a template.
* class is used to tell the compiler that we are creating a datatype or class template.
* identifier is used to tell the compiler what the name of the template would be. This is similar to naming a variable or macro and could be almost anything that we want to name it. In the above example it is data\_type.

Now, we would have to replace the data types everywhere in the function where we want to use a template data type which we have done like this in the above code:

data\_type max\_num (data\_type firstNumber, data\_type secondNumber) {

if (firstNumber > secondNumber) {

return firstNumber;

}

return secondNumber;

}

In the above code, we have replaced the actual data types with the template type that we created.

Now how this works internally is that during compilation, whenever the compiler comes across any function calls to a template function, it creates a new function based on the data types of the arguments.

The newly created function is then used for that particular function call.

In our above example, all the 4 functions that we replaced with the template function will be created by the compiler as all 4 of them are supposed to be used in the main method.

Note: the function body needs to be exactly same across multiple functions to replace them with a template function.

Note: For the identifier (template name), a shorter name like T is commonly used which we will be using henceforth. Use T as the template name instead of "data\_type" (the name that we used in the previous example).

Create a template function in the below code for sum of two numbers

#### Expected Output

12

5.85987

61510821416124

5.85987

#include <bits/stdc++.h>

using namespace std;

template<class T>

T sum(T firstNumber, T secondNumber) {

return firstNumber + secondNumber;

}

int main() {

int firstInt = 5, secondInt = 7;

float firstFloat = 3.14159, secondFloat = 2.71828;

long firstLong = 54362478632846, secondLong = 7148342783278;

double firstDouble = 2.71828182845, secondDouble = 3.1415926535;

cout << sum(firstInt, secondInt) << endl;

cout << sum(firstFloat, secondFloat) << endl;

cout << sum(firstLong, secondLong) << endl;

cout << sum(firstDouble, secondDouble) << endl;

return 0;

}

Now, let's take another case.

In the max\_num example, let's say that we had to find the max of firstLong and firstInt like this:

cout << max\_num(firstLong, firstInt) << endl;

Here, the return type is supposed to be long.

Our template function won't work because both the parameters have the same template data type in our template function whereas in the function call, we have long and int data as arguments. A function which can take these two as arguments can't be produced from this template function.

template <class T>

T max\_num (T firstNumber, T secondNumber) {

if (firstNumber > secondNumber) {

return firstNumber;

}

return secondNumber;

}

What can we do now?

Just like having a template data type T, we can have another data type, say, U. We can rewrite the above code as:

template <class T, class U>

T max\_num (T firstNumber, U secondNumber) {

if (firstNumber > secondNumber) {

return firstNumber;

}

return secondNumber;

}

Here, what is happening is that the first parameter will take data of type T, the second parameter will take data of type U and the function will return data of type T.

In our example with long and int, the compiler will use this template to create a function like this:

long max\_num (long firstNumber, int secondNumber) {

if (firstNumber > secondNumber) {

return firstNumber;

}

return secondNumber;

}

So now our problem is solved! Will this also work for (int, int), (float, float), (long, long), (double, double)?

The answer is yes. Here, T and U are independent of each other. T can be any data type, U can be any data type independent of each other. So it is possible to have T being int and independently U also being int.

Therefore, we can use the above template method for all the 5 use cases:

* (int, int)
* (float, float)
* (long, long)
* (double, double)
* (long, int)

As you can notice, the template is not just restricted to these 5 pairs of arguments. (int, long), (float, int) and any other pair of data types will be valid here.

Let's rewrite the previous code to use the template with T and U:

#include <bits/stdc++.h>

using namespace std;

template <class T, class U>

T max\_num (T firstNumber, U secondNumber) {

if (firstNumber > secondNumber) {

return firstNumber;

}

return secondNumber;

}

int main() {

int firstInt = 5, secondInt = 7;

float firstFloat = 3.14159, secondFloat = 2.71828;

long firstLong = 54362478632846, secondLong = 7148342783278;

double firstDouble = 2.71828182845, secondDouble = 3.1415926535;

cout << max\_num(firstInt, secondInt) << endl;

cout << max\_num(firstFloat, secondFloat) << endl;

cout << max\_num(firstLong, secondLong) << endl;

cout << max\_num(firstDouble, secondDouble) << endl;

//Newly added function calls. All of these are valid and will work

cout << max\_num(firstLong, firstInt) << endl;

cout << max\_num(firstInt, firstLong) << endl;

cout << max\_num(firstDouble, firstFloat) << endl;

cout << max\_num(firstDouble, firstInt) << endl;

}

Let's modify the sum function from above to work for 2 different types of parameters as well just like we did from max\_num.

#### Expected Output

12

5.85987

61510821416124

5.85987

54362478632851

1077567379

5.85987

7.71828

#include <bits/stdc++.h>

using namespace std;

template <class T, class U>

T sum (T firstNumber, U secondNumber) {

return firstNumber + secondNumber;

}

int main() {

int firstInt = 5, secondInt = 7;

float firstFloat = 3.14159, secondFloat = 2.71828;

long firstLong = 54362478632846, secondLong = 7148342783278;

double firstDouble = 2.71828182845, secondDouble = 3.1415926535;

cout << sum(firstInt, secondInt) << endl;

cout << sum(firstFloat, secondFloat) << endl;

cout << sum(firstLong, secondLong) << endl;

cout << sum(firstDouble, secondDouble) << endl;

//Newly added function calls. All of these are valid and will work

cout << sum(firstLong, firstInt) << endl;

cout << sum(firstInt, firstLong) << endl;

cout << sum(firstDouble, firstFloat) << endl;

cout << sum(firstDouble, firstInt) << endl;

return 0;

}

As we can see that the template can take both class and primitive data types (int, float, long, etc), it must be obvious that we can create a template where a class object is passed as a parameter as well.

##### Examples

template <class T>

T max\_obj (T firstObject, T secondObject) {

if (firstObject.some\_attribute > secondObject.some\_attribute) {

return firstObject;

}

return secondObject;

}

template <class T, class U>

int func (T firstObject, U secondObject) {

int result = 0;

//Some logic

return result;

}

##### FYI

In case of templates, we've been calling the functions like this:

cout << max\_num(firstInt, secondInt) << endl;

Here, we are relying on the compiler to deduce the type/class of the arguments and use the right function accordingly.

There is a way in which instead of relying on the compiler to deduce the type/class, we can specify it to the compiler directly.

For the below template function with only a single type/class:

template <class T>

T max\_num (T firstNumber, T secondNumber) {

if (firstNumber > secondNumber) {

return firstNumber;

}

return secondNumber;

}

We can do like this:

cout << max\_num<int>(firstInt, secondInt) << endl;

For the below template function with more than one template type/class:

template <class T, class U>

T max\_num (T firstNumber, U secondNumber) {

if (firstNumber > secondNumber) {

return firstNumber;

}

return secondNumber;

}

We can do like this for (int, int):

cout << max\_num<int, int>(firstInt, secondInt) << endl;

For (long, int):

cout << max\_num<long, int>(firstLong, firstInt) << endl;

This is just an FYI. In general, we might not have to do this.

# Object-Oriented Programming - Principles in C++: Polymorphism - II

## Class Templates

Let's create an Array class which contains an integer array and expose a few methods to work with the array. Go through the below code properly.

#include <bits/stdc++.h>

using namespace std;

class ArrayClass {

private:

int\* arr;

int current\_size = 0;

int max\_size;

public:

ArrayClass(int max\_size) {

this->max\_size = max\_size;

arr = new int[max\_size];

}

void insert(int element) {

arr[current\_size] = element;

current\_size++;

}

int get\_at(int index) {

return arr[index];

}

void print() {

for (int iterator = 0; iterator < current\_size; iterator++) {

cout << arr[iterator] << " ";

}

cout << endl;

}

};

int main() {

// your code goes here

ArrayClass array(10);

array.insert(5);

array.insert(6);

array.insert(7);

array.insert(8);

array.print();

cout << array.get\_at(2);

return 0;

}

Let's look at what is happening here:

* The Array class takes the max\_size in the constructor and creates an array with that size.
* The insert method takes an element and adds it at the last index (current\_size) and increments the current\_size.
* The get\_at methods takes an index and returns the element at that index.
* The print method prints the array.

Now, we have a class for int arrays. What if we want to create similar classes for arrays of other data types as well?

Similar to function templates, we can also have class templates. Just like function templates, class templates allow us to create template data types and use them inside the class to make the class generic.

Let's templatize the above code to create array of any data type (could be int, float as well as Employee or any other class).

Here, we have just replaced int with T wherever int was mentioned in the context of array element.

template<class T>

class ArrayClass {

private:

T\* arr;

int current\_size = 0;

int max\_size;

public:

ArrayClass(int max\_size) {

this->max\_size = max\_size;

arr = new T[max\_size];

}

void insert(T element) {

arr[current\_size] = element;

current\_size++;

}

T get\_at(int index) {

return arr[index];

}

void print() {

for (int iterator = 0; iterator < current\_size; iterator++) {

cout << arr[iterator] << " ";

}

cout << endl;

}

};

Everything looks fine, right?

There is one major issue. When we create the array object like this:

ArrayClass array(10);

The class won't be able to implicitly find out what is the data type of T. It could be int or float or anything. It will be unknown until a function call is made where T is present in parameters (In this case, insert method).

So, how do we fix it?

We have already learned in the previous section how to explicitly mention the data type while calling a template function like this:

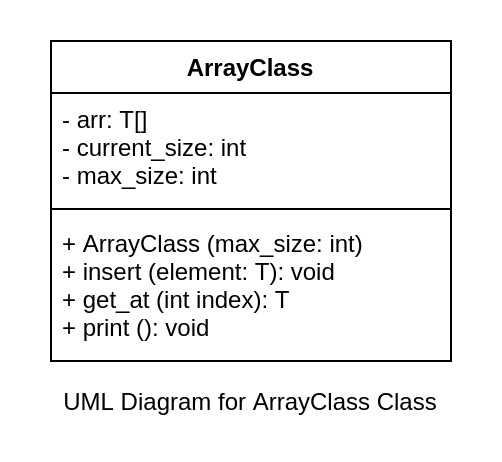
max\_num<int>(firstInt, secondInt)

The same thing can be done here as well like this:

* ArrayClass<int> array(10);
* ArrayClass<float> array2(10);
* ArrayClass<Employee> array3(10);

Class Templates are very commonly used in C++ Standard Template Library (STL).

The UML diagram for the above class would be:



Let's modify the below code to use templates

* Templatize the Stack class in the editor to allow the container array to work for non-int elements as well.
* Modify the object creation in the main method to work with the templatized Stack class.
* I've added comments wherever you need to templatize. Modify only those parts.
* Just focus on the templatizing part and the object creation part.
* You can ignore the logic for now.

#### Expected Output

0 1 0

5 0 1

0 1 0

0 1 0

3.14 0 1

0 1 0

1 0

w 0 1

1 0

#include <bits/stdc++.h>

using namespace std;

//Templatize the class

template<class T>

class Stack {

private:

T \*container; //Array for storing data. Needs to be templatized

int no\_of\_elements = 0;

int max\_size;

public:

Stack (int max\_size) {

this->max\_size = max\_size;

//Array initialization. Needs to be templatized

container = new T[max\_size];

}

bool empty() {

return no\_of\_elements == 0;

}

int size() {

return no\_of\_elements;

}

//Returns array element. Right now return\_type is int. Needs to be templatized

T top() {

if (no\_of\_elements == 0) {

return NULL;

}

return container[no\_of\_elements - 1];

}

//Takes element to be added to array. Right now param\_type is int. Needs to be templatized

void push(T element) {

if (no\_of\_elements == max\_size) {

return;

}

container[no\_of\_elements] = element;

no\_of\_elements++;

}

void pop() {

if (no\_of\_elements == 0) {

return;

}

no\_of\_elements--;

}

};

int main() {

// your code goes here

//Templatize this for int

Stack<int> stackInt(100);

cout << stackInt.top() << " " << stackInt.empty() << " " << stackInt.size() << endl;

stackInt.push(5);

cout << stackInt.top() << " " << stackInt.empty() << " " << stackInt.size() << endl;

stackInt.pop();

cout << stackInt.top() << " " << stackInt.empty() << " " << stackInt.size() << endl;

//Templatize this for float

Stack<float> stackFloat(100);

cout << stackFloat.top() << " " << stackFloat.empty() << " " << stackFloat.size() << endl;

stackFloat.push(3.14);

cout << stackFloat.top() << " " << stackFloat.empty() << " " << stackFloat.size() << endl;

stackFloat.pop();

cout << stackFloat.top() << " " << stackFloat.empty() << " " << stackFloat.size() << endl;

//Templatize this for char

Stack<char> stackChar(100);

cout << stackChar.top() << " " << stackChar.empty() << " " << stackChar.size() << endl;

stackChar.push('w');

cout << stackChar.top() << " " << stackChar.empty() << " " << stackChar.size() << endl;

stackChar.pop();

cout << stackChar.top() << " " << stackChar.empty() << " " << stackChar.size() << endl;

return 0;

}

# Object-Oriented Programming - Principles in C++: Polymorphism - II

## Templates

Both function templates and class templates allow us to write generic code as in functions and class operating on generic types. Generic programming is a style of computer programming in which code is written in terms of types to-be-specified-later as we have been doing in this chapter by adding a template data-type/class.

The C++ Standard Library includes the Standard Template Library or STL that provides a framework of templates for common data structures and algorithms.

***This type of polymorphism is also known as parametric polymorphism.***

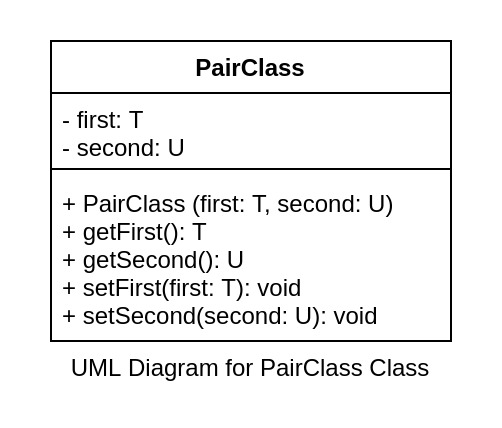
Just like in function overloading and operator overloading, the actual function/class to use is determined at compile time in case of templates as well and so templates is also a way to achieve compile-time polymorphism.

# Object-Oriented Programming - Principles in C++: Assessment - V

## Assessment - V

### Problem Statement:

Given a class Pair which is used to store data in pairs.



First and second can be of any data type irrespective of each other.

##### Examples

firstPair => (4, 2)

secondPair => (3.14, 2.718)

thirdPair => ("Hello", 3.14)

Implementation Details:

* All get and set functions are normal getters and setters.
* getFirst returns first
* getSecond returns second
* setFirst takes first as argument and assigns it to the attribute first.
* setSecond takes second as argument and assigns it to the attribute second.
* In the main method, create three pairs with the following values:
  + firstPair => (4, 2)
  + secondPair => (3.14, 2.718)
  + thirdPair => ("Hello", 3.14)
* Do not modify anything in the code after the "DO NOT MODIFY AFTER THIS" comment.

#### Expected Output

4 2

3.14 2.718

Hello 3.14

3 14

#include <bits/stdc++.h>

using namespace std;

template<class T, class U>

class PairClass{

private:

T first;

U second;

public:

PairClass(){}

PairClass(T first, U second){

this->first = first;

this->second = second;

}

T getFirst(){

return first;

}

U getSecond(){

return second;

}

void setFirst(T first){

this->first = first;

}

void setSecond(U second){

this->second = second;

}

};

int main() {

// your code goes here

PairClass<int,int> firstPair(4,2);

PairClass<double,double> secondPair(3.14,2.718);

PairClass<string,double> thirdPair("Hello",3.14);

//DO NOT MODIFY AFTER THIS

cout << firstPair.getFirst() << " " << firstPair.getSecond() << endl;

cout << secondPair.getFirst() << " " << secondPair.getSecond() << endl;

cout << thirdPair.getFirst() << " " << thirdPair.getSecond() << endl;

firstPair.setFirst(3);

firstPair.setSecond(14);

cout << firstPair.getFirst() << " " << firstPair.getSecond() << endl;

return 0;

}

**Output:**

4 2

3.14 2.718

Hello 3.14

4 2

# Object-Oriented Programming - Principles in C++: Polymorphism - III

## Pointer to derived class - I

**Polymorphism - III - Quiz 1**

Which property of OOP is denoted here?

Every CameraPhone is-a Phone

A. Abstration

B. Encapsulation

C. Inheritance

D. Polymorphism

Correct Answer is C.

In the following relationship, which of these will be the derived class?

Every CameraPhone is-a Phone

A. Only CameraPhone

B. Only Phone

C. Both of these

D. None of these

rect Answer is A.

In Inheritance, X extends Y.

Which of these is true?

A. X: Base Class, Y: Derived Class

B. X: Derived Class, Y: Base Class

C. X: Base Class, Y: Base Class

D. X: Derived Class, Y: Derived Class

Correct Answer is B.

Given an object of class Phone:

Phone phone\_var("Apple", "iPhone 11", 4, 64);

How do you access the property brand from this object?

A.

phone\_var['brand']

B.

phone\_var->brand

C.

phone\_var.brand

D.

phone\_var{brand}

Correct Answer is C.

Given a pointer to an object of class Phone:

Phone phone\_var("Apple", "iPhone 11", 4, 64);

Phone \*phone\_ptr = &phone\_var;

How do you access the property brand from this pointer variable?

A.

phone\_ptr['brand']

B.

phone\_ptr->brand

C.

phone\_ptr.brand

D.

phone\_ptr{brand}

Correct Answer is B.

We've previously learned that we can access class members through a pointer to an object like this:

Phone phone\_var("Apple", "iPhone 11", 4, 64);

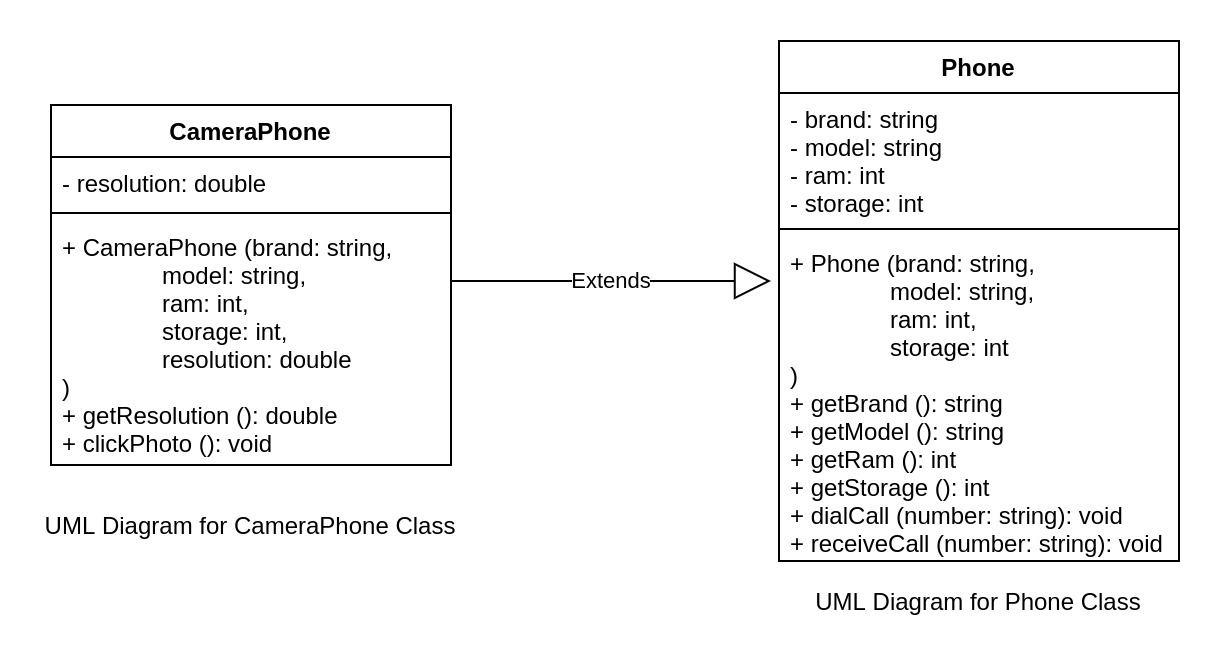
Phone \*phone\_ptr = &phone\_var;

cout << phone\_ptr->brand;

C++ allows us to point a pointer of base class to a derived class as well.

##### Example

Going back to our example of Phone and CameraPhone.



We already know that we can do something like this:

Phone phone\_var("Apple", "iPhone 11", 4, 64);

Phone \*phone\_ptr = &phone\_var;

cout << phone\_ptr->brand;

CameraPhone camera\_phone("Apple", "iPhone 11", 4, 64, 12);

CameraPhone \*camera\_phone\_ptr = &camera\_phone;

cout << camera\_phone\_ptr->brand;

What we can also do is this:

CameraPhone camera\_phone("Apple", "iPhone 11", 4, 64, 12);

Phone \*phone\_ptr = &camera\_phone;

cout << phone\_ptr->brand;

Here, we are creating a pointer of the base class Phone and pointing it to an object of the derived class CameraPhone.

Since, a CameraPhone is-a Phone and has all the accessible members of the Phone class, it makes sense for C++ to allow those class members to be available through a pointer of Phone class.

What this means is that when we do the following, we will have access only to the class members of CameraPhone that are inherited from the Phone class.

CameraPhone camera\_phone("Apple", "iPhone 11", 4, 64, 12);

cout << camera\_phone.brand << endl; //This is valid

cout << camera\_phone.resolution << endl; //This is valid

Phone \*phone\_ptr = &camera\_phone;

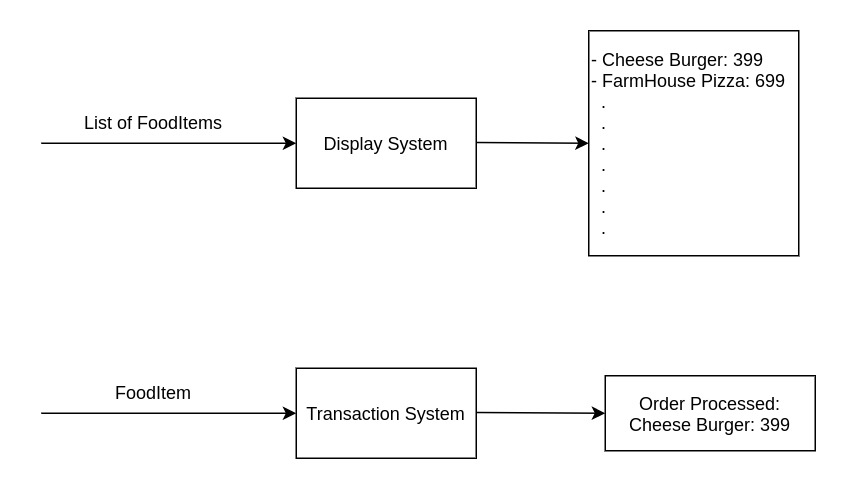
cout << phone\_ptr->brand << endl; //This is valid

cout << phone\_ptr->resolution << endl; **//This is not valid**

The pointer of the base class won't have access to any of the members of the derived class even when the object is of the derived class. Reason being that the pointer variable does not know anything about the derived class. It just knows about the class it is a pointer of, which is the base class.

But why would we want to use a pointer of the base class to point to an object of a derived class?

Let's say that you own an online food delivery service. You need to display a list of all the food items that the customers can order. You also have a transaction system to process the orders.



For your display and transaction systems, would you keep a list of Burger objects, list of Pizza objects, etc or would you keep a list of FoodItem objects?

If you keep a list of FoodItem objects, all your systems do not need to understand about the internals of different FoodItems. The internal detail can be abstracted out for the display and transaction systems. They only need to care about the name, image, price, restaurant name, etc. These systems do not care about the specific attributes or functions of a Pizza or a Burger, right?

Abstracting the actual class of the objects through pointer of the base class is generally useful when we have a list of objects of different types but having a common base class.

Now let's write a code for this:

* In the main function, create a CameraPhone object named camera\_phone with these attributes: ("Apple", "iPhone 11", 4, 64, 12);
* Create a pointer of class Phone named phone\_ptr
* Point it to the object camera\_phone
* Using phone\_ptr, print all the attributes of the variable camera\_phone that the pointer has access to. Print one-per-line.

#### Expected Output

Apple

iPhone 11

4

64

#include <bits/stdc++.h>

using namespace std;

class Phone {

string brand;

string model;

int ram;

int storage;

public:

Phone (string brand, string model, int ram, int storage) {

this->brand = brand;

this->model = model;

this->ram = ram;

this->storage = storage;

}

string getBrand() {

return this->brand;

}

string getModel() {

return this->model;

}

int getRam() {

return this->ram;

}

int getStorage() {

return this->storage;

}

void dialCall (string number) {

cout << "Calling " << number << " from " << brand << ":" << model << "\n";

}

void receiveCall (string number) {

cout << "Receiving call from " << number << " on " << brand << ":" << model << "\n";

}

};

class CameraPhone: public Phone {

double resolution;

public:

CameraPhone(string brand, string model, int ram, int storage, double resolution): Phone(brand, model, ram, storage) {

this->resolution = resolution;

}

double getResolution() {

return this->resolution;

}

void clickPhoto () {

cout << "Clicking photo on a " << resolution << " MP " << getBrand() << ":" << getModel() << "\n";

}

};

int main() {

// your code goes here

CameraPhone camera\_phone("Apple", "iPhone 11", 4, 64, 12);

Phone \*phone\_ptr = &camera\_phone;

cout << phone\_ptr->getBrand() <<"\n";

cout << phone\_ptr->getModel() <<"\n";

cout << phone\_ptr->getRam() <<"\n";

cout << phone\_ptr->getStorage() <<"\n";

return 0;

}

# Object-Oriented Programming - Principles in C++: Polymorphism - III

## Pointer to derived class - II

**Polymorphism - III - Quiz 2**

What will be the output of the below code:

class BaseClass {

public:

void print() {

cout << "Base Class" << endl;

}

};

class DerivedClass: BaseClass {

public:

void print() {

cout << "Derived Class" << endl;

}

};

int main() {

BaseClass base;

DerivedClass derived;

base.print();

derived.print();

return 0;

}

A.

Base Class

Base Class

B.

Base Class

Derived Class

C.

Derived Class

Base Class

D.

Derived Class

Derived Class

Correct Answer is B.

Given that CameraPhone is derived from Phone class, what will be the output of the following code?

CameraPhone's constructor takes 5 parameters: brand, model, ram, storage, resolution

CameraPhone camera\_phone("Apple", "iPhone 11", 4, 64, 12);

Phone \*phone\_ptr = &camera\_phone;

cout << phone\_ptr->brand;

A. Apple

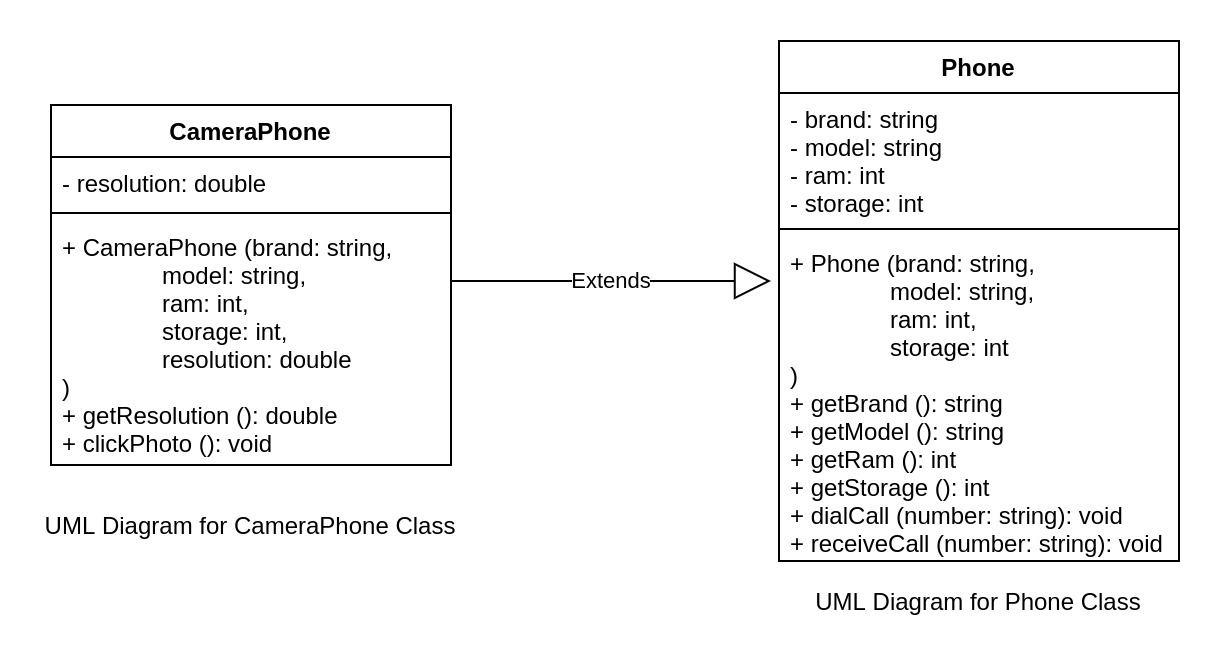
B. Address of camera\_phone

C. Address of brand

D. Syntax Error as phone\_ptr is assigned address of CameraPhone

Correct Answer is A.

Going back to our example of Phone and CameraPhone.



Let's say that we have an e-commerce website which lists phones and show details about them.

As we've learned in the previous section, we can abstract out the actual class through a pointer of the base class. Here, we can have the Phone/CameraPhone objects through Phone pointers like this:

Phone phone("Nokia", "110", 1, 1);

CameraPhone camera\_phone("Apple", "iPhone", 4, 64, 12);

Phone \*phone\_ptr\_1 = &phone;

Phone \*phone\_ptr\_2 = &camera\_phone;

Complete the below quiz based on the above UML diagram.

**Polymorphism - III - Quiz 3**

For the e-commerce website, we want to display the details of a phone with the attributes of each phone separated by a space. We can achieve this by adding a display method in both Phone and CameraPhone class.

Which of the following would you add to the Phone class?

A.

void display () {

cout << brand << " " << model << " " << ram << " " << storage << endl;

}

B.

void display () {

cout << brand << " " << model << " " << ram << " " << storage << " " << resolution << endl;

}

C.

void display () {

cout << resolution << " " << model << " " << ram << " " << storage << endl;

}

D.

void display () {

cout << resolution << endl;

}

Correct Answer is A.

Which of the following would you ideally add to the CameraPhone class?

A.

void display () {

cout << brand << " " << model << " " << ram << " " << storage << endl;

}

B.

void display () {

cout << brand << " " << model << " " << ram << " " << storage << " " << resolution << endl;

}

C.

void display () {

cout << getBrand() << " " << getModel() << " " << getRam() << " " << getStorage() << endl;

}

D.

void display () {

cout << getBrand() << " " << getModel() << " " << getRam() << " " << getStorage() << " " << getResolution() << endl;

}

Correct Answer is D.

Based on the display functions in the previous questions, what will be the output of the following code?

Phone phone("Nokia", "110", 1, 1);

CameraPhone camera\_phone("Apple", "iPhone", 4, 64, 12);

phone.display();

camera\_phone.display();

A.

Nokia 110 1 1

Apple iPhone 4 64

B.

Nokia 110 1 1 0

Apple iPhone 4 64 12

C.

Nokia 110 1 1

Apple iPhone 4 64 12

D.

Nokia 110 1 1 12

Apple iPhone 4 64

Correct Answer is C.

Based on the display functions in the previous questions, what will be the output of the following code?

Phone phone("Nokia", "110", 1, 1);

CameraPhone camera\_phone("Apple", "iPhone", 4, 64, 12);

Phone \*phone\_ptr = &phone;

CameraPhone \*camera\_phone\_ptr = &camera\_phone;

phone\_ptr->display();

camera\_phone\_ptr->display();

A.

Nokia 110 1 1

Apple iPhone 4 64

B.

Nokia 110 1 1 0

Apple iPhone 4 64 12

C.

Nokia 110 1 1

Apple iPhone 4 64 12

D.

Nokia 110 1 1 12

Apple iPhone 4 64

orrect Answer is C.

C

D

Correct Answer is A.

Which of the following would you ideally add to the CameraPhone class?

A.

void display () {

cout << brand << " " << model << " " << ram << " " << storage << endl;

}

B.

void display () {

cout << brand << " " << model << " " << ram << " " << storage << " " << resolution << endl;

}

C.

void display () {

cout << getBrand() << " " << getModel() << " " << getRam() << " " << getStorage() << endl;

}

D.

void display () {

cout << getBrand() << " " << getModel() << " " << getRam() << " " << getStorage() << " " << getResolution() << endl;

}

A

B

C

D

Correct Answer is D.

Based on the display functions in the previous questions, what will be the output of the following code?

Phone phone("Nokia", "110", 1, 1);

CameraPhone camera\_phone("Apple", "iPhone", 4, 64, 12);

phone.display();

camera\_phone.display();

A.

Nokia 110 1 1

Apple iPhone 4 64

B.

Nokia 110 1 1 0

Apple iPhone 4 64 12

C.

Nokia 110 1 1

Apple iPhone 4 64 12

D.

Nokia 110 1 1 12

Apple iPhone 4 64

A

B

C

D

Correct Answer is C.

Based on the display functions in the previous questions, what will be the output of the following code?

Phone phone("Nokia", "110", 1, 1);

CameraPhone camera\_phone("Apple", "iPhone", 4, 64, 12);

Phone \*phone\_ptr = &phone;

CameraPhone \*camera\_phone\_ptr = &camera\_phone;

phone\_ptr->display();

camera\_phone\_ptr->display();

A.

Nokia 110 1 1

Apple iPhone 4 64

B.

Nokia 110 1 1 0

Apple iPhone 4 64 12

C.

Nokia 110 1 1

Apple iPhone 4 64 12

D.

Nokia 110 1 1 12

Apple iPhone 4 64

A

B

C

D

Correct Answer is C.

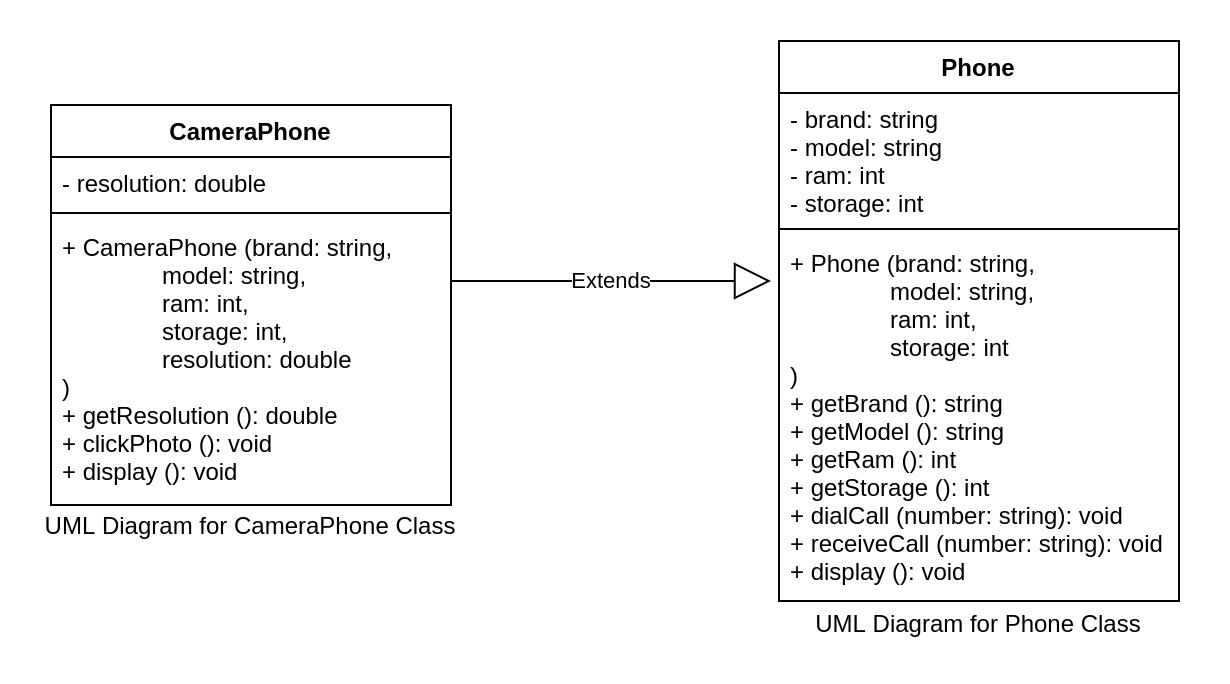
**Your Score**

4

out of 4

Restart

Based on the above quiz, we add display function to both Phone and CameraPhone.



The display function in the Phone class would be:

void display () {

cout << getBrand() << " " << getModel() << " " << getRam() << " " << getStorage() << endl;

}

The display function in the CameraPhone class would be:

void display () {

cout << getBrand() << " " << getModel() << " " << getRam() << " " << getStorage() << " " << getResolution() << endl;

}

We can directly call the display method on the object of the respective classes (Phone or CameraPhone) wherever we want.

**Polymorphism - III - Quiz 4**

Given that CameraPhone is derived from Phone class, what will be the output of the following code?

CameraPhone's constructor takes 5 parameters: brand, model, ram, storage, resolution

CameraPhone camera\_phone("Apple", "iPhone 11", 4, 64, 12);

Phone \*phone\_ptr = &camera\_phone;

cout << phone\_ptr->brand;

A. Apple

B. Address of camera\_phone

C. Address of brand

D. Syntax Error as phone\_ptr is assigned address of CameraPhone

Correct Answer is A.

Given that CameraPhone is derived from Phone class, what will be the output of the following code?

CameraPhone's constructor takes 5 parameters: brand, model, ram, storage, resolution

CameraPhone camera\_phone("Apple", "iPhone 11", 4, 64, 12);

Phone \*phone\_ptr = &camera\_phone;

cout << phone\_ptr->resolution;

A. 12

B. Address of camera\_phone

C. Address of resolution

D. Error as resolution is not an attribute of Phone class

Correct Answer is D.

If we use pointer of base class, what will be the output when we do the following?

Phone phone("Nokia", "110", 1, 1);

CameraPhone camera\_phone("Apple", "iPhone", 4, 64, 12);

Phone \*phone\_ptr\_1 = &phone;

Phone \*phone\_ptr\_2 = &camera\_phone;

phone\_ptr\_1->display();

phone\_ptr\_2->display();

We would expect that the display function of Phone will be called through phone\_ptr\_1 and of CameraPhone through phone\_ptr\_2 and that the output should be:

Nokia 110 1 1

Apple iPhone 4 64 12

But this won't happen. The display function of Phone will be called through both phone\_ptr\_1 and phone\_ptr\_2 and the actual output would be:

Nokia 110 1 1

Apple iPhone 4 64

As we've previously seen while trying to do phone\_ptr->resolution:

The pointer of the base class won't have access to any of the members of the derived class even when the object is of the derived class. Reason being that the pointer variable does not know anything about the derived class. It just knows about the class it is a pointer of, which is the base class.

Here, phone\_ptr\_2 will not have access to the display method of the CameraPhone class and so the display method of the Phone class will be called.

***So, how to use the functions of the respective class even when a pointer of the base class is used to reference the object?***

***Through “virtual functions”***! We will learn everything about virtual functions in the next section.

If you notice carefully that what we want to achieve here is having the same function name with different implementations based on the context which means that we are trying to achieve polymorphism.

# Object-Oriented Programming - Principles in C++: Polymorphism - III

## Virtual Function

**Polymorphism - III - Quiz 5**

Given that CameraPhone is derived from Phone class, what will be the output of the following code?

CameraPhone's constructor takes 5 parameters: brand, model, ram, storage, resolution

CameraPhone camera\_phone("Apple", "iPhone 11", 4, 64, 12);

Phone \*phone\_ptr = &camera\_phone;

cout << phone\_ptr->brand;

A. Apple

B. Address of camera\_phone

C. Address of brand

D. Syntax Error as phone\_ptr is assigned address of CameraPhone

Correct Answer is A.

Given that CameraPhone is derived from Phone class, what will be the output of the following code?

CameraPhone's constructor takes 5 parameters: brand, model, ram, storage, resolution

CameraPhone camera\_phone("Apple", "iPhone 11", 4, 64, 12);

Phone \*phone\_ptr = &camera\_phone;

cout << phone\_ptr->resolution;

A. 12

B. Address of camera\_phone

C. Address of resolution

D. Error as resolution is not an attribute of Phone class

Correct Answer is D.

The display function in the Phone class would be:

void display () {

cout << getBrand() << " " << getModel() << " " << getRam() << " " << getStorage() << endl;

}

The display function in the CameraPhone class would be:

void display () {

cout << getBrand() << " " << getModel() << " " << getRam() << " " << getStorage() << " " << getResolution() << endl;

}

Based on the above display functions, what will be the output of the following code?

Phone phone("Nokia", "110", 1, 1);

CameraPhone camera\_phone("Apple", "iPhone", 4, 64, 12);

phone.display();

camera\_phone.display();

A.

Nokia 110 1 1

Apple iPhone 4 64

B.

Nokia 110 1 1 0

Apple iPhone 4 64 12

C.

Nokia 110 1 1

Apple iPhone 4 64 12

D.

Nokia 110 1 1 12

Apple iPhone 4 64

Correct Answer is C.

Given that CameraPhone is derived from Phone class and based on the display functions of the previous questions, what will be the output when we do the following?

Phone phone("Nokia", "110", 1, 1);

CameraPhone camera\_phone("Apple", "iPhone", 4, 64, 12);

Phone \*phone\_ptr = &phone;

CameraPhone \*camera\_phone\_ptr = &camera\_phone;

phone\_ptr->display();

camera\_phone\_ptr->display();

A.

Nokia 110 1 1

Apple iPhone 4 64

B.

Nokia 110 1 1 0

Apple iPhone 4 64 12

C.

Nokia 110 1 1

Apple iPhone 4 64 12

D.

Nokia 110 1 1 12

Apple iPhone 4 64

Correct Answer is C.

Given that CameraPhone is derived from Phone class and based on the display functions of the previous questions, what will be the output when we do the following?

Phone phone("Nokia", "110", 1, 1);

CameraPhone camera\_phone("Apple", "iPhone", 4, 64, 12);

Phone \*phone\_ptr\_1 = &phone;

Phone \*phone\_ptr\_2 = &camera\_phone;

phone\_ptr\_1->display();

phone\_ptr\_2->display();

A.

Nokia 110 1 1

Apple iPhone 4 64

B.

Nokia 110 1 1 0

Apple iPhone 4 64 12

C.

Nokia 110 1 1

Apple iPhone 4 64 12

D.

Nokia 110 1 1 12

Apple iPhone 4 64

Correct Answer is A.

In the previous sections, we learned that:

* We can access class members through an object using the dot operator.
* We can use pointers to access class elements by using the arrow operator.
* We can create pointers of base class and point it to an object of derived class.
* We can use these pointers to access class members of derived class that are inherited from the base class.
* We cannot use these pointers to access class members of derived class that are not inherited from the base class.
* Calling a method present in both base class and derived class through a pointer of base class will result in the method from base class being called.

Now, our problem here is that if we want to abstract out the actual class with the base class, we lose access to methods that we might want to use. Given a list of phones, we would want the display method to print the value based on the display method in the respective class.

To solve this problem, what we would want to do is override the function in the base class with the function in the derived class. For a particular function in the base class, if there is no other implementation present in the derived class then there is no problem as the base function is the one that needs to be called anyway. What we need here is the flexibility to add a different function implementation in any of the derived classes and use that function if it has been overridden in the derived class. This is known as function overriding.

Now we know what is the problem and what we want to do. Let's see how to do it.

C++ has a way to override a function through the virtual keyword.

Based on the use case if we want a base class function to be overridden we can add the virtual keyword and then any derived class would be able to override it. Function overriding with the virtual keyword is used quite often when we are writing code using OOP in C++.

##### Syntax

class ClassName {

.

.

.

**virtual** return\_type function\_name(function\_parameter\_1, function\_parameter\_2, …..) {

//function logic

}

}

##### Example

class Phone {

.

.

.

virtual void display () {

cout << getBrand() << " " << getModel() << " " << getRam() << " " << getStorage() << endl;

}

};

class CameraPhone {

.

.

.

void display () {

cout << getBrand() << " " << getModel() << " " << getRam() << " " << getStorage() << " " << getResolution() << endl;

}

}

Note that I have added the virtual keyword in the display method of class Phone.

Now, we will get the desired output when we do this:

Phone phone("Nokia", "110", 1, 1);

CameraPhone camera\_phone("Apple", "iPhone", 4, 64, 12);

Phone \*phone\_ptr\_1 = &phone;

Phone \*phone\_ptr\_2 = &camera\_phone;

phone\_ptr\_1->display();

phone\_ptr\_2->display();

#### Expected Output

Nokia 110 1 1

Apple iPhone 4 64 12

Let's see it in action. Complete the tasks mentioned in the comments and then hit run to check if we are getting the expected output (mentioned above).

#include <bits/stdc++.h>

using namespace std;

class Phone {

string brand;

string model;

int ram;

int storage;

public:

Phone (string brand, string model, int ram, int storage) {

this->brand = brand;

this->model = model;

this->ram = ram;

this->storage = storage;

}

string getBrand() {

return this->brand;

}

string getModel() {

return this->model;

}

int getRam() {

return this->ram;

}

int getStorage() {

return this->storage;

}

void dialCall (string number) {

cout << "Calling " << number << " from " << brand << ":" << model << "\n";

}

void receiveCall (string number) {

cout << "Receiving call from " << number << " on " << brand << ":" << model << "\n";

}

// Task: Add the virtual function display with the implementation

virtual void display(){

cout << getBrand() << " " << getModel() << " " << getRam() << " " << getStorage() << endl;

}

};

class CameraPhone: public Phone {

double resolution;

public:

CameraPhone(string brand, string model, int ram, int storage, double resolution): Phone(brand, model, ram, storage) {

this->resolution = resolution;

}

double getResolution() {

return this->resolution;

}

void clickPhoto () {

cout << "Clicking photo on a " << resolution << " MP " << getBrand() << ":" << getModel() << "\n";

}

// Task: Add the function display with the implementation

void display(){

cout << getBrand() << " " << getModel() << " " << getRam() << " " << getStorage() << " " << getResolution() << endl;

}

};

int main () {

Phone phone("Nokia", "110", 1, 1);

CameraPhone camera\_phone("Apple", "iPhone", 4, 64, 12);

Phone \*phone\_ptr\_1 = &phone;

Phone \*phone\_ptr\_2 = &camera\_phone;

phone\_ptr\_1->display();

phone\_ptr\_2->display();

return 0;

}

# Object-Oriented Programming - Principles in C++: Polymorphism - III

## Runtime Polymorphism

In case of function overriding, the actual function that is supposed to be called is not known during compile-time and can only be deduced while the code is being executed (at run-time). Hence, it is used to achieve runtime polymorphism.

This is also known as dynamic dispatch where dispatch means finding the correct function to call. It is done dynamically (at run-time) hence the prefix "dynamic".

The compiler does some work which is then used to select the correct function during run-time. Since the concept has more to do with internal implementation and not application-level implementation, you might not be able to retain it after a while. Focus on understanding the concept and revise it when you need to.

You can read more about the internal workings in [this amazing article](https://pabloariasal.github.io/2017/06/10/understanding-virtual-tables) if you want to.

# Object-Oriented Programming - Principles in C++: Polymorphism - III

## Abstract Classes

We've seen in the course different ways to achieve abstraction. We have also learned about abstracting the actual class of an object by its base class. There are certain use cases where there is no point in creating objects of the base class.

##### Example

Base Class: Animal

Derived Classes: Human, Lion, Snake, Fish, Eagle, etc.

or

Base Class: ElectronicDevice

Derived Classes: Phone, Laptop, Refrigerator, Television, etc.

or

Base Class: Employee

Derived Classes: IndividualContributor, Manager, CEO, etc.

***Here, we would never want to create an object of the base class as the base class can't exist as a whole. It is always something which is a generalization of something whole/concrete (derived classes). Here, we would always create objects of the derived classes.***

*These types of classes where the only purpose of the class is to provide abstraction to the derived classes are known as* ***abstract classes***.

In case of abstract classes, many methods of the base class will be used as is through objects of the derived classes. If you notice carefully, there will be certain methods in the abstract classes which will always be overridden by the derived classes and would not have any implementation of its own.

##### Example

Animal will be an abstract class. There will be certain methods like move, speak, etc which won't have any implementation in the Animal class. What we can do is not even put these methods in the Animal class. What will be the problem if we do that?

***The purpose of having an abstract class is to abstract out the actual class of the objects.*** We've previously learned that we cannot access the members of the derived class through a pointer of the base class unless that member has a definition in the base class as well. So, we will have to keep the method in the abstract class as well. ***We can anyway have an empty method*** definition in the base class like this:

virtual void move() {

}

***Note that the virtual keyword is important here as the method is supposed to be overridden.***

***As you can see that till now we have not put any restriction on the abstract class. Creating objects of an abstract class should not be allowed as it cannot exist as an actual whole entity but we are unable to restrict it right now. So it is not actually an abstract class right now.***

***C++ allows us to create an abstract class through pure virtual functions***. ***Pure virtual functions are functions which have no implementation and should always be implemented by the derived classes***. Didn't we already do that in the above code?

The above code still had an empty implementation. This is how we can rewrite using pure virtual functions:

virtual void move() = 0;

**Here,** ***move is a pure virtual function*** **and** ***the Animal class would become an abstract class.*** Now, if we try to create an object of Animal class, we would get an error because the function move won't be defined.

The syntax for creating a pure virtual function is:

class ClassName {

.

.

.

virtual return\_type function\_name(parameter\_1, parameter\_2, .....) = 0;

}

### Example

Please go through the code properly and complete the tasks mentioned in the code comments.

#### Expected Output

Walk

Crawl

Fly

Walk

Fly

#include <bits/stdc++.h>

using namespace std;

class Animal {

public:

void eat(string food) {

cout << "Eating " << food;

}

virtual void move() = 0;

};

class Human: public Animal {

public:

void move() {

cout << "Walk";

}

};

class Snake: public Animal {

public:

void move() {

cout << "Crawl";

}

};

class Eagle: public Animal {

public:

void move() {

cout << "Fly";

}

};

int main() {

// your code goes here

//

// Note that we cannot create Animal objects now as Animal is an abstract class

// The following code will give error if uncommented.

// Task: Uncomment the next line, run and see the error. Comment again after that.

// Animal animal;

Human human\_1;

Snake snake\_1;

Eagle eagle\_1;

Human human\_2;

Eagle eagle\_2;

//Creating an array of Animal pointers

Animal \*animals[5];

//Assigned addresses of human\_1 and eagle\_1 to 0th and 1st index of animals

animals[0] = &human\_1;

animals[1] = &snake\_1;

animals[2] = &eagle\_1;

animals[3] = &human\_2;

animals[4] = &eagle\_2;

//Task: Assign addresses of eagle\_1, human\_2 and eagle\_2 to 2nd, 3rd and 4th index of animals

//Task: Hit run and see what gets printed and if it is what you expected

for (int i = 0; i < 5; i++) {

animals[i]->move();

cout << endl;

}

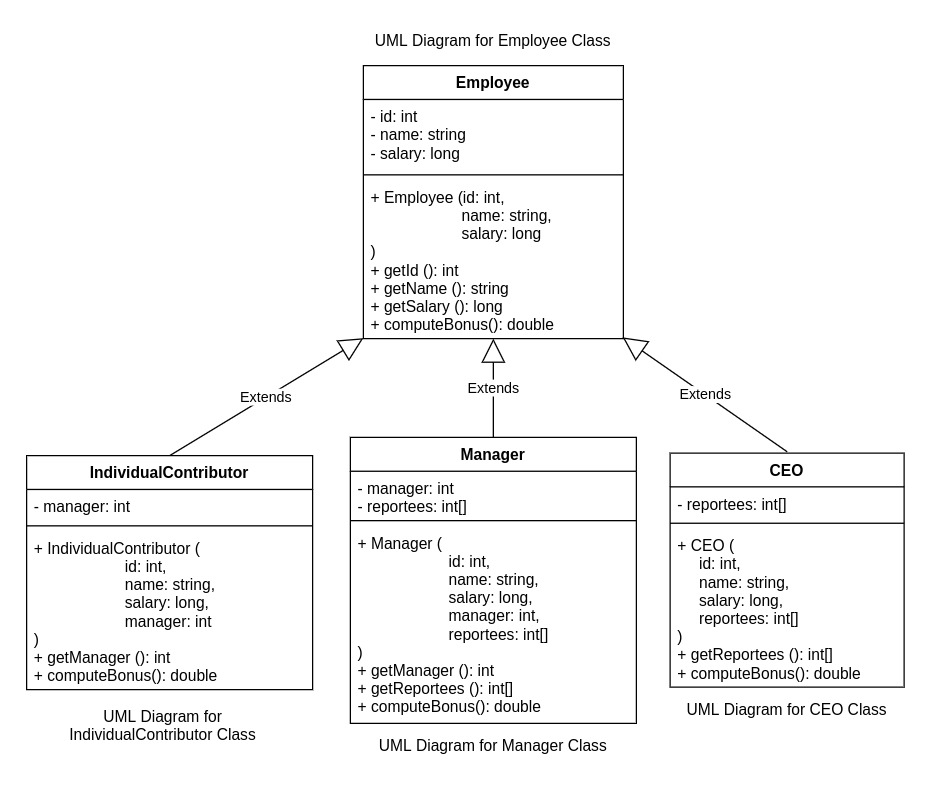
return 0;

}

# Object-Oriented Programming - Principles in C++: Assessment - VI & VII

## Assessment - VI

### Problem Statement:



Update this code to use the following logic.

* Add the addresses of all the objects to the employees array like this:

employees[0] = &lester

* Allow function overriding of the computeBonus method of Employee class

#### Expected Output

10500000.00

5500000.00

1150000.00

950000.00

10000.00

10000.00

#include <bits/stdc++.h>

using namespace std;

class Employee {

int id;

string name;

long salary;

public:

Employee(int id, string name, long salary) {

this->id = id;

this->name = name;

this->salary = salary;

}

int getId() {

return id;

}

string getName() {

return name;

}

long getSalary() {

return salary;

}

virtual double computeBonus() {

return (salary\*10)/100.0;

}

};

class IndividualContributor: public Employee {

int manager;

public:

IndividualContributor(int id, string name, long salary, int manager): Employee(id, name, salary) {

this->manager = manager;

}

int getManager() {

return manager;

}

double computeBonus() {

return (getSalary()\*10)/100.0;

}

};

class Manager: public Employee {

int manager;

int \*reportees;

public:

Manager(int id, string name, long salary, int manager, int \*reportees): Employee(id, name, salary) {

this->manager = manager;

this->reportees = reportees;

}

int getManager() {

return manager;

}

int\* getReportees() {

return reportees;

}

double computeBonus() {

if (getSalary() > 3000000) {

return 150000 + (getSalary()\*20)/100.0;

} else {

return (getSalary()\*25)/100.0;

}

}

};

class CEO: public Employee {

int \*reportees;

public:

CEO(int id, string name, long salary, int \*reportees): Employee(id, name, salary) {

this->reportees = reportees;

}

int\* getReportees() {

return reportees;

}

double computeBonus() {

return 500000 + (getSalary()\*50)/100.0;

}

};

int main() {

// your code goes here

int ceoReportees[] = {1, 2};

int terryReportees[] = {4};

int tyrellReportees[] = {5};

CEO lester(0, "Lester Moore", 20000000, ceoReportees);

CEO phillip(1, "Phillip Price", 10000000, ceoReportees);

Manager terry(2, "Terry Colby", 5000000, 0, terryReportees);

Manager tyrell(3, "Tyrell Wellick", 4000000, 0, tyrellReportees);

IndividualContributor elliot(4, "Elliot Alderson", 100000, 2);

IndividualContributor angela(5, "Angela Moss", 100000, 3);

Employee \*employees[6];

//Assign addresses of the above employees (ceos, managers and ics) to the employees array

employees[0] = &lester;

employees[1] = &phillip;

employees[2] = &terry;

employees[3] = &tyrell;

employees[4] = &elliot;

employees[5] = &angela;

cout << fixed << setprecision(2);

for (int i = 0; i < 6; i++) {

cout << employees[i]->computeBonus() << endl;

}

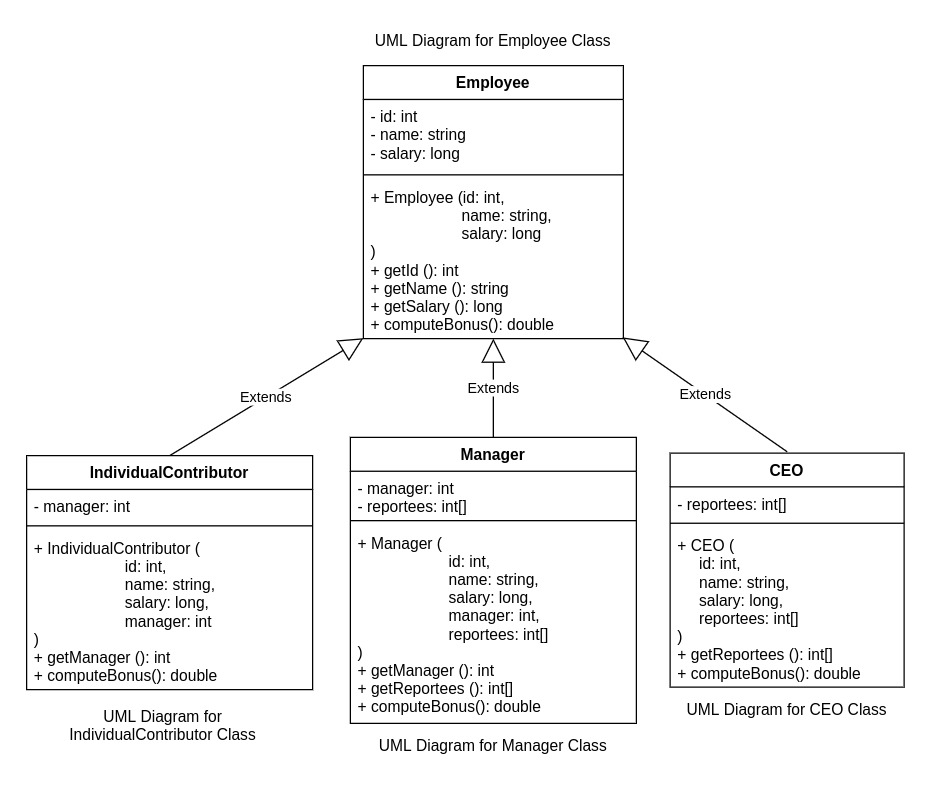
return 0;

}

# Object-Oriented Programming - Principles in C++: Assessment - VI & VII

## Assessment - VII

### Problem Statement:



Since Employee class only acts as a base class and it does not make sense to create an Employee object directly, we can make it an abstract class.

* Update this code to use the following logic.
* Add the addresses of all the objects to the employees array like this:

employees[0] = &lester

* Convert the Employee class into an abstract class

#### Expected Output

10500000.00

5500000.00

1150000.00

950000.00

10000.00

10000.00

#include <bits/stdc++.h>

using namespace std;

class Employee {

int id;

string name;

long salary;

public:

Employee(int id, string name, long salary) {

this->id = id;

this->name = name;

this->salary = salary;

}

int getId() {

return id;

}

string getName() {

return name;

}

long getSalary() {

return salary;

}

virtual double computeBonus() {

return (salary\*10)/100.0;

}

};

class IndividualContributor: public Employee {

int manager;

public:

IndividualContributor(int id, string name, long salary, int manager): Employee(id, name, salary) {

this->manager = manager;

}

int getManager() {

return manager;

}

double computeBonus() {

return (getSalary()\*10)/100.0;

}

};

class Manager: public Employee {

int manager;

int \*reportees;

public:

Manager(int id, string name, long salary, int manager, int \*reportees): Employee(id, name, salary) {

this->manager = manager;

this->reportees = reportees;

}

int getManager() {

return manager;

}

int\* getReportees() {

return reportees;

}

double computeBonus() {

if (getSalary() > 3000000) {

return 150000 + (getSalary()\*20)/100.0;

} else {

return (getSalary()\*25)/100.0;

}

}

};

class CEO: public Employee {

int \*reportees;

public:

CEO(int id, string name, long salary, int \*reportees): Employee(id, name, salary) {

this->reportees = reportees;

}

int\* getReportees() {

return reportees;

}

double computeBonus() {

return 500000 + (getSalary()\*50)/100.0;

}

};

int main() {

// your code goes here

int ceoReportees[] = {1, 2};

int terryReportees[] = {4};

int tyrellReportees[] = {5};

CEO lester(0, "Lester Moore", 20000000, ceoReportees);

CEO phillip(1, "Phillip Price", 10000000, ceoReportees);

Manager terry(2, "Terry Colby", 5000000, 0, terryReportees);

Manager tyrell(3, "Tyrell Wellick", 4000000, 0, tyrellReportees);

IndividualContributor elliot(4, "Elliot Alderson", 100000, 2);

IndividualContributor angela(5, "Angela Moss", 100000, 3);

Employee \*employees[6];

//Assign addresses of the above employees (ceos, managers and ics) to the employees array

employees[0] = &lester;

employees[1] = &phillip;

employees[2] = &terry;

employees[3] = &tyrell;

employees[4] = &elliot;

employees[5] = &angela;

cout << fixed << setprecision(2);

for (int i = 0; i < 6; i++) {

cout << employees[i]->computeBonus() << endl;

}

return 0;

}

# Object-Oriented Programming - Principles in C++: Additional Concepts

## Composition

Given the relationship between a base class and derived class.

DerivedClass \_\_\_\_ BaseClass

Fill in the blank.

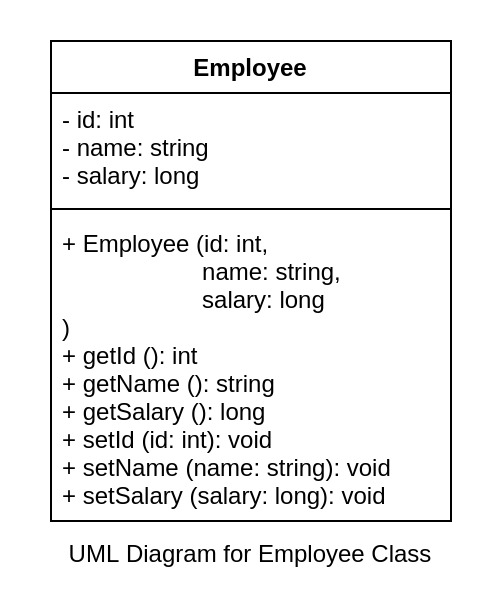
A. is-a

B. has-a

C. is-not-a

D. uses-a

Correct Answer is A.



In this example, we can see that the class Employee is composed of an int id, a string name and a long salary**. This property where objects or data types are combined to create more objects (from classes) is known as *composition.***

Let's look at how these are related:

* Employee has-a id
* Employee has-a name
* Employee has-a salary

Therefore, the relationship in composition is "has-a".

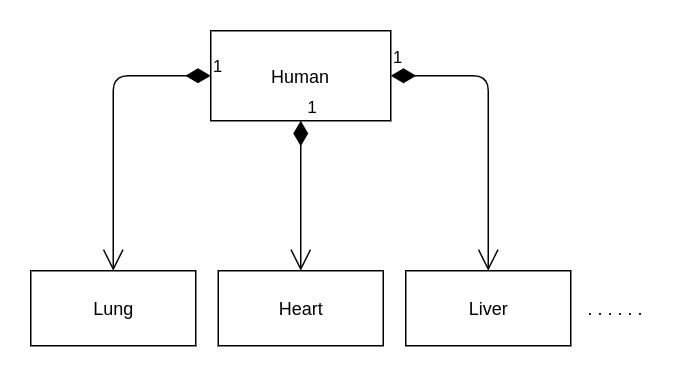
Let's look at a few more examples of composition:

* Human has-a heart. Here, heart is an object of class Heart.
* Car has-a engine. Here, engine is an object of class Engine.

In case of composition, the attribute has no existence outside the class. If a class object is destroyed, all its attributes also get destroyed with it.

In fact the class knows about all its attributes but the attribute has no idea of the class it is a part of.

The UML Diagram of a composition relationship between multiple classes looks like this:



Here, the parent class Human is composed of multiple child classes. They are represented using an arrow between them where the parent side has a filled black diamond whereas the child class has a normal arrow.

There could be one or more instances of a child class. In the above example, there will be two attributes of type Lung.

The above UML diagram would look something like this in code:

class Human {

private:

Lung left\_lung;

Lung right\_lung;

Heart heart;

.

.

.

};

# Object-Oriented Programming - Principles in C++: Additional Concepts

## FYI

Please note that most of the concepts covered beyond this point are important but not frequently used unless you work with C++ in a production setting.

Since you would not be using it regularly, it won't be easy for you to retain it for long. So the idea here is to focus on understanding the concepts and its use case. Do not focus on retaining the exact syntax unless you plan to use it in the near future.

If understand the concepts covered, it should be easy to Google and start using when you actually need to use them.

# Object-Oriented Programming - Principles in C++: Additional Concepts

## Destructor

**Destructor - Quiz 1**

Which of these functions is used to create an object of a class?

A. Creator

B. Director

C. Constructor

D. Destructor

Correct Answer is C.

Which of these is a default constructor of class Phone?

A.

Phone() {

//some initialization

}

B.

Phone(string brand, string model, int ram, int storage) {

//some initialization

}

C.

Phone constructor() {

//some initialization

}

D.

Phone Phone() {

//some initialization

}

Correct Answer is A.

C

D

Correct Answer is C.

Which of these is a default constructor of class Phone?

A.

Phone() {

//some initialization

}

B.

Phone(string brand, string model, int ram, int storage) {

//some initialization

}

C.

Phone constructor() {

//some initialization

}

D.

Phone Phone() {

//some initialization

}

A

B

C

D

Correct Answer is A.

**Your Score**

2

out of 2

Restart

We have been creating variables and objects throughout the course. As we know that all variables and objects take up memory. If we keep on running different code with objects and variables, won't the entire RAM get filled up?

It usually happens when we run a lot of programs together (especially Chrome). But it does not happen if we run just a few programs even if we do it for years. The memory should get filled even in that case, right?

The reason why this does not happen is that variable or object is destroyed as soon as it gets out of scope and the memory gets released.

##### Example

int square (int num) {

int sqr = num\*num;

return sqr;

}

Here, the variables num and sqr also take up memory when the function square is called. As soon as these variables get out of scope (when we get out of the function), the memory held by these variables get released.

All the variables declared inside main method is destroyed when we exit out of the main method (the program execution completes).

All the variables declared inside any code block like if or for is also destroyed as soon as that code block ends.

Similar to constructors that are used to create objects of a class, we also have destructors which are used to destroy an object. A destructor is automatically called when the object is supposed to be destroyed (i.e., when it goes out of scope) and the memory where the object is stored gets released.

A destructor is defined similar to a constructor but with a tilde(~) before the function.

For class Phone, the destructor will be defined like this:

~Phone() {

}

***This is the default destructor which gets created by the compiler if no other destructor is present in the class.***

A destructor cannot have any parameters as passing data to it does not make sense. Moreover, we do not need to call a destructor explicitly as it is automatically called when the object goes out of scope.

We can create our own destructor like this:

~Phone() {

//some destruction logic

}

In general, we do not need to create our own destructor but we may have to create when we have any pointers or any memory that we have dynamically allocated like creating an array inside the class.

# Object-Oriented Programming - Principles in C++: Additional Concepts

## Friend Function

**Friend Function - Quiz 1**

If we want to expose members of a class only in that class and its derived classes and not outside these, which access specifier should we use for that member?

A. public

B. private

C. protected

D. hidden

Correct Answer is C.

If we want to expose members of a class only in that class and not outside that, which access specifier should we use for that member?

A. public

B. private

C. protected

D. hidden

Correct Answer is B.

If a class has some protected class members, which of the following can access those members?

A. Only members of the same class

B. Only members of the same class and its base class

C. Only members of the same class and its derived classes

D. Any class

orrect Answer is C.

If a class has some private class members, which of the following can access those members?

A. Members of the same class

B. Members of its base class

C. Members of its derived classes

D. Any class

Correct Answer is A.

**Your Score**

4

out of 4

Restart

*We know that a private class member is accessible only inside that class. What if we want to allow some non-member function to access the private members of a class?*

**C++ allows us to do that through the friend keyword.** ***We can make a non-member function as a friend of a class. This will allow the friend function to have the same access to the class members as enjoyed by other members of that class.***

##### Example

Let's say we've a class named Phone and there is a function outside the class named reviewPhone. Now, since the reviewPhone is outside the Phone class, it won't have access to the private members of class Phone. But if we make reviewPhone as a friend of class Phone, it'll have the same access to the class members of Phone as any other function inside the Phone class.

Let's see how to make a function as a class's friend. Please go through the code and then run it.

The syntax of declaring a friend function is:

friend return\_type function\_name(comma\_separated\_data\_type\_of\_parameters);

##### Example

friend int sum(int, int, int);

Note that it is not mandatory to have the class as part of the parameters.

##### Example

class Phone {

private:

string brand;

string model;

public:

//Here, we do not have a constructor

//hence default constructor will be used for object creation

//Making createApplePhone as a friend of the class

friend Phone createApplePhone(string);

};

Phone createApplePhone(string model) {

Phone phone;

phone.brand = "Apple";

phone.model = model;

return phone;

}

Please note that we should use friend functions very rarely and only in very specific cases. Try not to overuse it.

**Object-Oriented Programming - Principles in C++: Additional Concepts**

**Operator Overloading - II**

As we've learned in Operator Overloading - I, we can use operators on objects as well by overloading the operators for a class. We will take the example of ComplexNumber which has two private attributes/properties: real and imaginary.

class ComplexNumber {

private:

int real;

int imaginary;

public:

ComplexNumber (int real, int imaginary) {

this->real = real;

this->imaginary = imaginary;

}

int getRealPart () {

return real;

}

int getImaginaryPart () {

return imaginary;

}

}

ComplexNumber operator + (ComplexNumber firstNumber, ComplexNumber secondNumber) {

int real\_part = firstNumber.getRealPart() + secondNumber.getRealPart();

int imaginary\_part = firstNumber.getImaginaryPart() + secondNumber.getImaginaryPart();

return ComplexNumber(real\_part, imaginary\_part);

}

What if the getRealPart and getImaginaryPart functions were not public?

class ComplexNumber {

private:

int real;

int imaginary;

public:

ComplexNumber (int real, int imaginary) {

this->real = real;

this->imaginary = imaginary;

}

void print () {

cout << this->real << " + i" << this->imaginary;

}

};

In that case, we would not be able to get the real and imaginary attributes of the objects of that class.

We've previously learned that there are two types of functions which have access to a class's private properties:

* Class Members or Functions of the same class
* Friend Functions of that class

We can use both of these ways to access private members of a class to overload an operator.

Let's first convert the above code to use friend functions.

class ComplexNumber {

private:

int real;

int imaginary;

public:

ComplexNumber (int real, int imaginary) {

this->real = real;

this->imaginary = imaginary;

}

void print () {

cout << this->real << " + i" << this->imaginary;

}

friend ComplexNumber operator + (ComplexNumber, ComplexNumber);

};

ComplexNumber operator + (ComplexNumber firstNumber, ComplexNumber secondNumber) {

int real\_part = firstNumber.real + secondNumber.real;

int imaginary\_part = firstNumber.imaginary + secondNumber.imaginary;

return ComplexNumber(real\_part, imaginary\_part);

}

int main() {

// your code goes here

ComplexNumber c1(1, 5), c2(6, 7);

ComplexNumber c3 = c1 + c2;

c3.print();

return 0;

}

Here, we have made the operator function as a friend function of the ComplexNumber class.

Let's look at how to do operator overloading using member function in the next section.

# Object-Oriented Programming - Principles in C++: Additional Concepts

## Operator Overloading - III

Let's look at how to do operator overloading using member function.

If we have an object, you know that a member function will have access to all the attributes of that particular object.

##### Example

In the above code, print method has access to both real and imaginary.

Let's create another method in the ComplexNumber class named plus. It should also have access to attributes of that object.

class ComplexNumber {

private:

int real;

int imaginary;

public:

ComplexNumber (int real, int imaginary) {

this->real = real;

this->imaginary = imaginary;

}

void print () {

cout << this->real << " + i" << this->imaginary;

}

ComplexNumber plus (ComplexNumber secondNumber) {

int real\_part = this->real + secondNumber.real;

int imaginary\_part = this->imaginary + secondNumber.imaginary;

return ComplexNumber(real\_part, imaginary\_part);

}

};

Here, we can call the plus method like this:

ComplexNumber c1(1, 5), c2(6, 7);

ComplexNumber c3 = c1.plus(c2); //This is equivalent to c1 + c2

Given that operators are also functions, we can replace plus with "operator +" in the above example like this:

class ComplexNumber {

private:

int real;

int imaginary;

public:

ComplexNumber (int real, int imaginary) {

this->real = real;

this->imaginary = imaginary;

}

void print () {

cout << this->real << " + i" << this->imaginary;

}

ComplexNumber operator + (ComplexNumber secondNumber) {

int real\_part = this->real + secondNumber.real;

int imaginary\_part = this->imaginary + secondNumber.imaginary;

return ComplexNumber(real\_part, imaginary\_part);

}

};

This is how we do operator overloading using member function.

Congrats! You've successfully completed the OOP principles in C++ course.