**Object-Oriented Programming - Basics in C++: Introduction**

**Classes & Objects**

You are doing the "Object-Oriented Programming - Basics in C++" course on a website named workat.tech on Google Chrome on your Lenovo laptop while sitting on a black chair and your laptop kept on a wooden table in a room of your house.

Why am I telling you all this?

I am doing this to tell you how everything in the world is an object including ourselves. In the above statement, let us take a look at the different objects.

* 'Object-Oriented Programming - Basics in C++' course is an object of type course
* 'workat.tech' is an object of type website
* 'Google Chrome' is an object of type browser
* 'Lenovo laptop' is an object of type laptop
* 'Black chair' is an object of type chair
* 'Wooden table' is an object of type table
* 'Your room' is an object of type room
* 'Your house' is an object of type house
* 'You' are an object of type human

Note that everything in the real-world is an object of some type. The type can be very specific like 'your wooden table' could be said to be of type table or could also be generic and be said to be of type furniture.  
'You' could be said to be an object of type Human or could also be said to be of type Organism.  
'Your Lenovo laptop' could of type Laptop or generically "Electronic Device".

Let's discuss this specificity at a later stage. At this stage, let's give a name to the type that we are talking about.

These types are known as classes. A class is a definition or a blueprint of objects of the same type.

We all are objects created from the blueprint of a Human class. Your Lenovo laptop is created from the blueprint of a Laptop class.

Which of the following classes best identifies a Tesla car?

Camera

Vehicle

Fruit

Animal

Which of the following classes best identifies a Tesla car?

Camera

Animal

Fruit

Car

Which of these can be said to be an object of class Country?

Bill Gates

Microsoft

India

Earth

Which of these can be said to be an object of class Organization?

Bill Gates

Microsoft

India

Earth

Let's say if we were to create mobile phones, what are some of the different attributes that a phone might have.

* Brand
* Model
* RAM
* Storage
* Processor
* Screen Size
* Operating System
* Color
* Weight

Let's also take a look at the different functions that the phone needs to allow the user to do:

* Dial Call
* Receive Call
* Send Message
* Open App

Now the attributes and the functions together make up the Mobile class (blueprint for any mobile phone).

Which of these is used to define a definition or blueprint of an object?

OOP

Function

Class \/

Attribute

Which of these make up a class?

A. Only Function

B. Only Attribute

C. Both Function and Attribute

D. None of the above

A

B

C \/

D

Which of these cannot be an attribute of type Table?

A. Number of legs

B. Number of eyes \/

C. Material

D. Color

Now, that we understand classes in terms of objects. Let's see how we can define objects in terms of classes.

An object is an instance of a class. There can be multiple copies created from a blueprint. Each copy is known as an instance.

Examples:

* You are an instance of class Human.
* Your laptop is an instance of class Laptop.

A class contains the definition of certain attributes and functions. Basically all the things that denote a particular type of thing.

An object contains the specific values of those attributes. The object can be used to perform certain functions as defined by the class.

The data and functions within a class are called members of the class.

There is no way to interact directly with a class as they are logical entities. We can interact with objects as they are real or abstract entities either present in the real world or in the digital world.

# Object-Oriented Programming - Basics in C++: Introduction

## Object Oriented Programming

Object-oriented programming (OOP) is a programming paradigm based on the concept of objects. In OOP, computer programs are designed out of multiple objects that interact with one another.

What we have learned in previous courses is known as procedural programming as it was based on writing procedures (functions/methods) to access or operate on data. The primary focus of procedural languages is functions.

In OOP, data and the functions that operate on them are bound together in classes. The primary focus of OOP languages is data (objects).

C++ which is an object-oriented programming language was mainly created to allow object orientation to the C language.

**Object-Oriented Programming - Basics in C++: Introduction**

**Abstraction**

When we are using our mobile phone, we know how to change the volume or click a photo or lock/unlock the phone. But what we don't know is how it internally works. The internal implementation is abstracted out from us.

Abstraction is a fundamental concept in computer science and software development where the implementation is hidden from the end user.

While using this website and running code on the IDE, you are actually interacting with an abstraction layer where you do not actually know or have to deal with how the code is executed internally. You engage with the website just through a button.

While learning to code, we actually created many abstraction layers without actually knowing that we are creating abstractions.

These are few of the scenarios where you must have used abstraction while learning to code:

* Creating functions and using them. To the caller, the function is supposed to do a task. The caller does not care about the internal implementation of that function.
* We used scan and print functionalities without caring about how it works internally.
* We even wrote code without knowing how the compiler understands the code and how the compiled code is understood by the computer.

Apart from writing well-structured code based on entities, OOP helps us achieve a good amount of abstraction as well in our code. We will cover multiple concepts which help us achieve abstraction in the OOP courses.

# Object-Oriented Programming - Basics in C++: Classes & Objects

## Class Definition

Which of these is not a valid in-built data type in C++?

int

float

char

decimal

What type of data is represented by int data type?

Fraction

Integer \/

Words

Lists

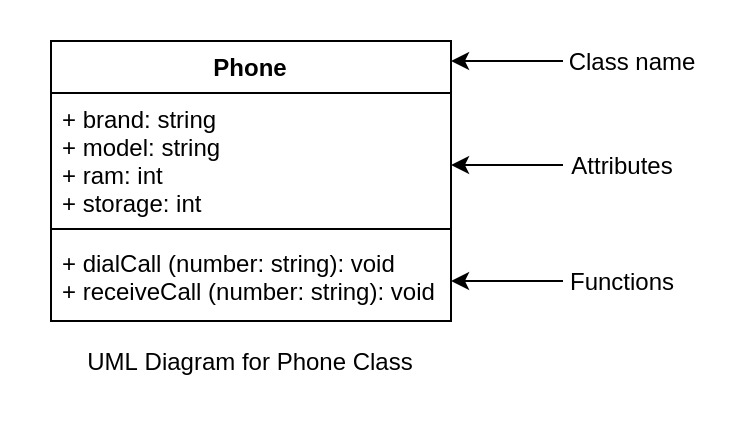
Similar to how int, float, char, etc are in-built data types used to represent different types of data, a class is a user-defined data type used to represent some type of data.

***Unlike primitive data types like int, float, etc, classes can have different attributes to represent the data and certain functions to operate on that data.***

***The attributes and functions of a class are also known as the class members.***

Let's look at an example class.  
What are the different attributes of a phone?  
What are the different functions a phone can perform?

##### Example



The above diagram denotes the class Phone with its attributes and functions. **This type of diagram is known as *UML (Unified Modeling Language) Class Diagram.*** *It is used to visualize a class and its relationships.*Note that we have picked up a limited set of attributes and functions of a phone for the sake of simplicity.

P.S.: We've used "string" instead of character array in this course. string is a class in C++ used to represent sequence of characters (commonly known as a string).

Q1. Based on the above UML diagram, which of these is not an attribute of Phone class?

brand

model

ram

dialCall \/

Q2. Based on the above UML diagram, which of these is not a member of Phone class?

brand

model

dialCall

sendMessage

Q3. How do you declare a string variable in C++?

A.

int brand;

B.

char brand;

C.

string brand;

D.

brand string;

Q4.Which of these is a valid method?

A.

void dialCall(string number) {

return "Calling " + number + "\n";

}

B.

void dialCall(string number) {

cout << "Calling " << number << "\n";

}

C.

void dialCall(void number) {

cout << "Calling " << number << "\n";

}

D.

string dialCall(string number) {

cout << "Calling " << number << "\n";

}

Correct Answer is B.

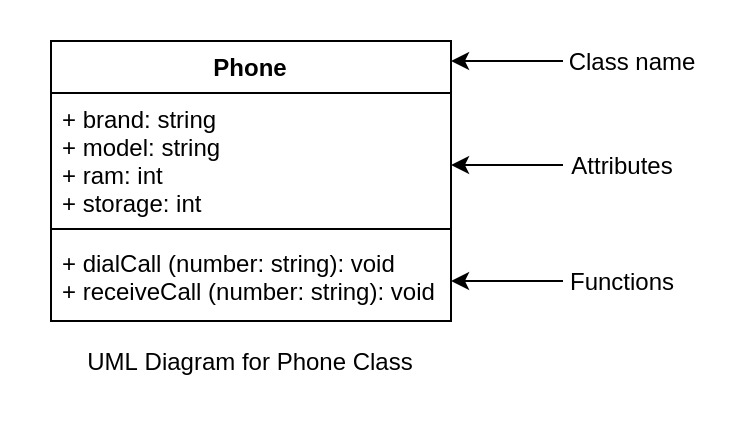
void dialCall(string number) {

cout << "Calling " << number << "\n";

}

Every other code has some mistake either in return\_type or return\_value or parameter data\_type.

Now let's look at how to define the Phone class through a C++ code.



class Phone {

public:

string brand;

string model;

int ram;

int 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";

}

};

Here, we are creating a class named Phone with all of the members mentioned above.

Apart from the public keyword, the above code should be self-explanatory based on intuition.

We will discuss the significance of public in a later section.

***Here, attributes are just like variables and class methods are just like any other method/function. They are just bound together in a single entity: the class.***

Please note that we can access all the attributes of the class in the class methods.

Example:

//Here dialCall method can access any of the class attributes (brand, model, etc)

void dialCall(string number) {

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

}

For now, let us look at the generic structure of a class in C++:

class ClassName {

private:

data\_type attribute\_name;

data\_type attribute\_name;

.

.

.

return\_type function\_name( parameters ) {

//Some logic

}

.

.

.

public:

data\_type attribute\_name;

data\_type attribute\_name;

.

.

.

return\_type function\_name( parameters ) {

//Some logic

}

.

.

.

};

Here, we declare the class with:

* A "class" keyword followed by the class name
* A code block with the following data
  + private/public keyword (To be discussed in a later section)
  + Class members (attributes and functions)
* ***Semicolon (;) at the end of the class code block***

Please do not forget to add the semicolon at the end of the class code block. It is a common mistake that beginners make and end up getting a compilation error.

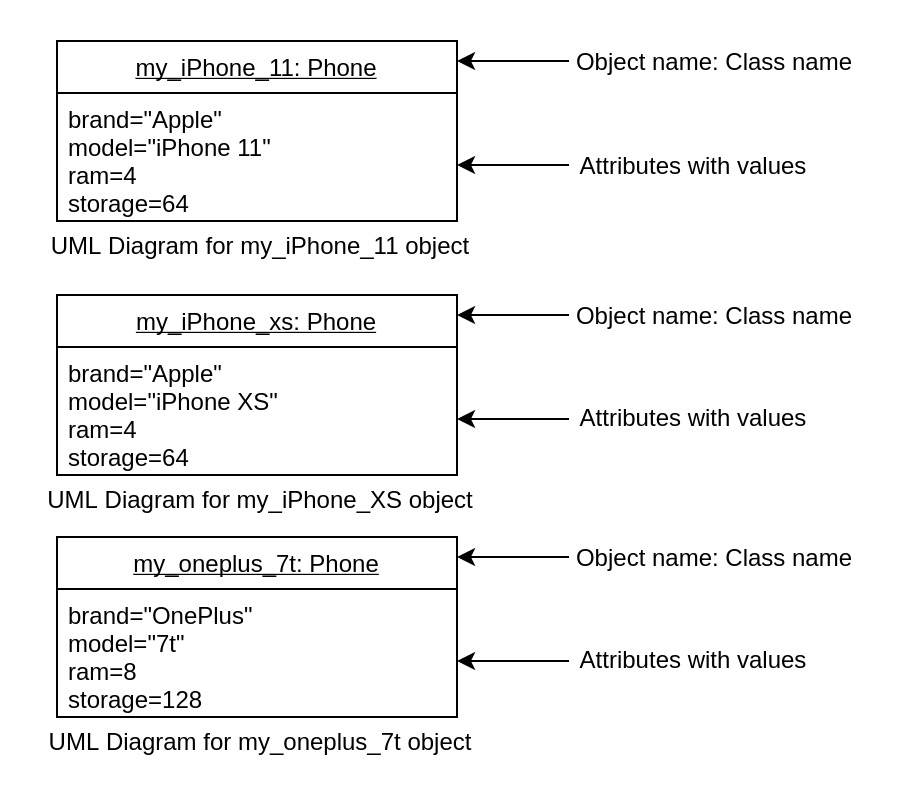
**Object-Oriented Programming - Basics in C++: Classes & Objects**

**Creating objects**

We have learned the relationship between classes and objects in the previous sections. We also learned how to define a class in C++.

Let's look at how to create objects from that class.

These are sample UML object diagrams for objects created through the phone class.



How to create these objects in C++?

#include <bits/stdc++.h>

using namespace std;

class Phone {

public:

string brand;

string model;

int ram;

int 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";

}

};

int main() {

// your code goes here

Phone my\_iPhone\_11;

Phone my\_iPhone\_xs;

Phone my\_oneplus\_7;

return 0;

}

Similar to how we use in-built data types (int, float, etc) to create variables of those types, we do the same for an object (a variable of a user-defined class) as well. We can create as many objects as we want similar to how we can create any number of int variables.

int num;

Phone my\_iPhone\_11;

Class is analogous to a data type and an object is analogous to a variable of that data type.

As you can see that we are not setting any value to the objects created in the above code. Let's look at how to do it.

We can access class members using the dot (.) operator on the object.

Example:

int main() {

Phone my\_iPhone\_11;

my\_iPhone\_11.brand = "Apple";

my\_iPhone\_11.model="iPhone 11";

my\_iPhone\_11.ram=4;

my\_iPhone\_11.storage=64;

cout << my\_iPhone\_11.brand << " " << my\_iPhone\_11.model << " " << my\_iPhone\_11.ram << " " << my\_iPhone\_11.storage << "\n";

my\_iPhone\_11.dialCall("9732130450");

my\_iPhone\_11.receiveCall("9732130450");

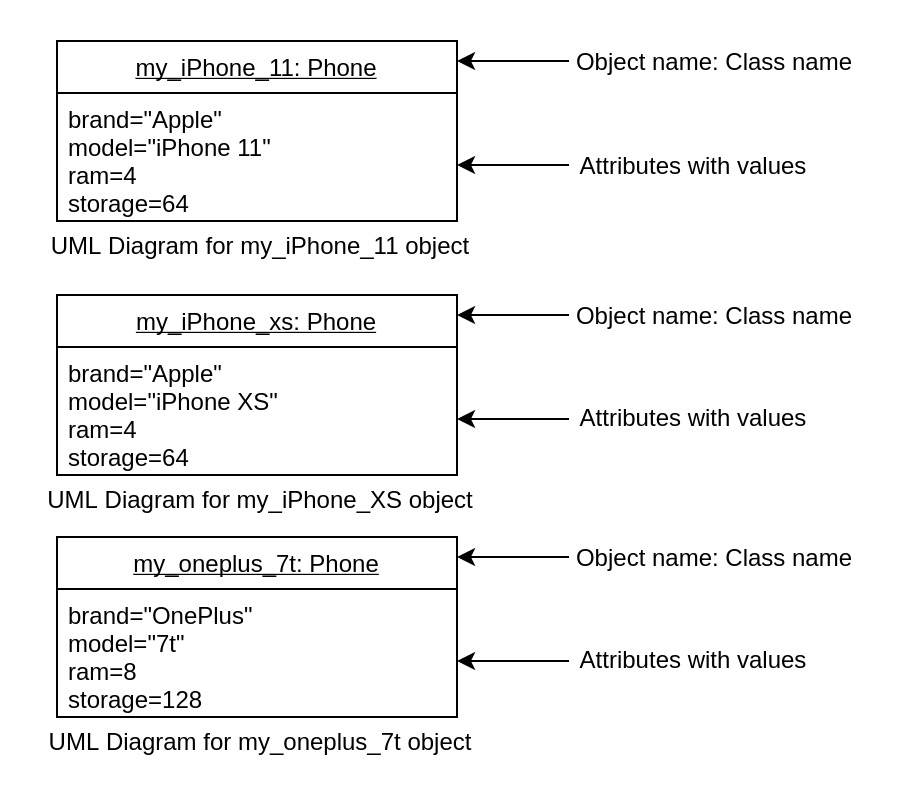
return 0;

}

Here, we are assigning the value "Apple" to my\_iPhone\_11.brand just like we assign a value to a variable.

Then, we are accessing the value of my\_iPhone\_11.brand just like we access the value of a variable.

Let's write some code.

* Create Phone class based on the structure defined above
* In the main method, create 3 objects of type Phone  
  
* Assign the values based on the UML diagram
* These objects should have the values as defined below
* For each object:
  + Print all the attributes of that objects.
  + Call dialCall and receiveCall from that object with 9732130450 as the phone number.

#### Expected Output

Apple iPhone 11 4 64

Calling 9732130450 from Apple:iPhone 11

Receiving call from 9732130450 on Apple:iPhone 11

Apple iPhone XS 4 64

Calling 9732130450 from Apple:iPhone XS

Receiving call from 9732130450 on Apple:iPhone XS

OnePlus 7t 8 128

Calling 9732130450 from OnePlus:7t

Receiving call from 9732130450 on OnePlus:7t

#include <bits/stdc++.h>

using namespace std;

class Phone {

public:

string brand;

string model;

int ram;

int 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";

}

};

int main() {

// class for Apple iPhone 11

Phone my\_iPhone\_11;

my\_iPhone\_11.brand = "Apple";

my\_iPhone\_11.model="iPhone 11";

my\_iPhone\_11.ram=4;

my\_iPhone\_11.storage=64;

cout << my\_iPhone\_11.brand << " " << my\_iPhone\_11.model << " " << my\_iPhone\_11.ram << " " << my\_iPhone\_11.storage << "\n";

my\_iPhone\_11.dialCall("9732130450");

my\_iPhone\_11.receiveCall("9732130450");

// class for Apple iPhone XS

Phone iPhone\_XS;

iPhone\_XS.brand = "Apple";

iPhone\_XS.model="iPhone XS";

iPhone\_XS.ram=4;

iPhone\_XS.storage=64;

cout << iPhone\_XS.brand << " " << iPhone\_XS.model << " " << iPhone\_XS.ram << " " << iPhone\_XS.storage << "\n";

iPhone\_XS.dialCall("9732130450");

iPhone\_XS.receiveCall("9732130450");

// class for OnePlus 7t

Phone OnePlus\_7t;

OnePlus\_7t.brand = "OnePlus";

OnePlus\_7t.model="7t";

OnePlus\_7t.ram=8;

OnePlus\_7t.storage=128;

cout << OnePlus\_7t.brand << " " << OnePlus\_7t.model << " " << OnePlus\_7t.ram << " " << OnePlus\_7t.storage << "\n";

OnePlus\_7t.dialCall("9732130450");

OnePlus\_7t.receiveCall("9732130450");

}

**Object-Oriented Programming - Basics in C++: Classes & Objects**

**Constructor**

**OOP Basics Introduction - Quiz 5**

Which of these is a valid statement for creating an object of class Phone?

A.

Phone my\_iPhone\_11;

B.

my\_iPhone\_11 Phone;

C.

my\_iPhone\_xs Phone;

D.

my\_oneplus\_7 Phone;

Correct Answer is A.

Phone my\_iPhone\_11;

Here, Phone is a class and my\_iPhone\_11 is an object of Phone.

Which of these is a valid way of assigning a value to an attribute of the object in C++?

A.

my\_iPhone\_11 = { brand: "Apple" };

B.

my\_iPhone\_11.brand => "Apple";

C.

my\_iPhone\_11.brand = "Apple";

D.

my\_iPhone\_11.brand == "Apple"

Correct Answer is C.

What is the name used to identify phoneNumber here?

void dialCall (string phoneNumber) {

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

}

Macros

Parameters

Arguments

Constants

Correct Answer is B.

Till now we have learned how to create an object and assign values to object properties using the dot operator.

How the object creation works is through constructors.

A constructor is a special type of class function with the name that is the same as the class name. It is used for creating objects of the class it belongs to.

The constructor for the phone class would be defined like this:

class Phone {

public:

string brand;

string model;

int ram;

int storage;

Phone () {

//any initialization logic

}

Phone (string brandValue, string modelValue, int ramValue, int storageValue) {

brand = brandValue;

model = modelValue;

ram = ramValue;

storage = storageValue;

}

void dialCall(string number) {

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

}

void receiveCall(string number) {

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

}

};

In the first constructor, we have created a method with no parameter. We can have any initialization logic that we want to do for that particular object.

In the second constructor, we have created a method with multiple parameters. What we are doing here is assigning the received values to the object. Let's see it by creating an object without such a constructor.

Phone my\_iPhone\_11;

my\_iPhone\_11.brand = "Apple";

my\_iPhone\_11.model="iPhone 11";

my\_iPhone\_11.ram=4;

my\_iPhone\_11.storage=64;

Now let's look at the same example of object creation with the same data through such a constructor.

Phone my\_iPhone\_11 = Phone ("Apple", "iPhone 11", 4, 64);

or

Phone my\_iPhone\_11 ("Apple", "iPhone 11", 4, 64);

Note that we do not mention the return type in the constructor function. It is a special function used to create an object of the same class in which the constructor is defined.

We also do not need to return anything from the constructor function, unlike other functions. It internally returns the same object which it is initializing in that method.

A constructor with parameters is known as a parameterized constructor.

A constructor without any parameters is known as the default constructor.

Why is it known as the "default" constructor?

It is automatically generated in the absence of any other constructors.

In the previous section, we were able to create an object without a constructor.

This is how we assigned values to class attributes earlier:

Phone my\_iPhone\_11;

my\_iPhone\_11.brand = "Apple";

my\_iPhone\_11.model="iPhone 11";

my\_iPhone\_11.ram=4;

my\_iPhone\_11.storage=64;

What is happening here is that when we are creating the object, the default constructor is called internally and the object is created through that constructor.

Phone my\_iPhone\_11;

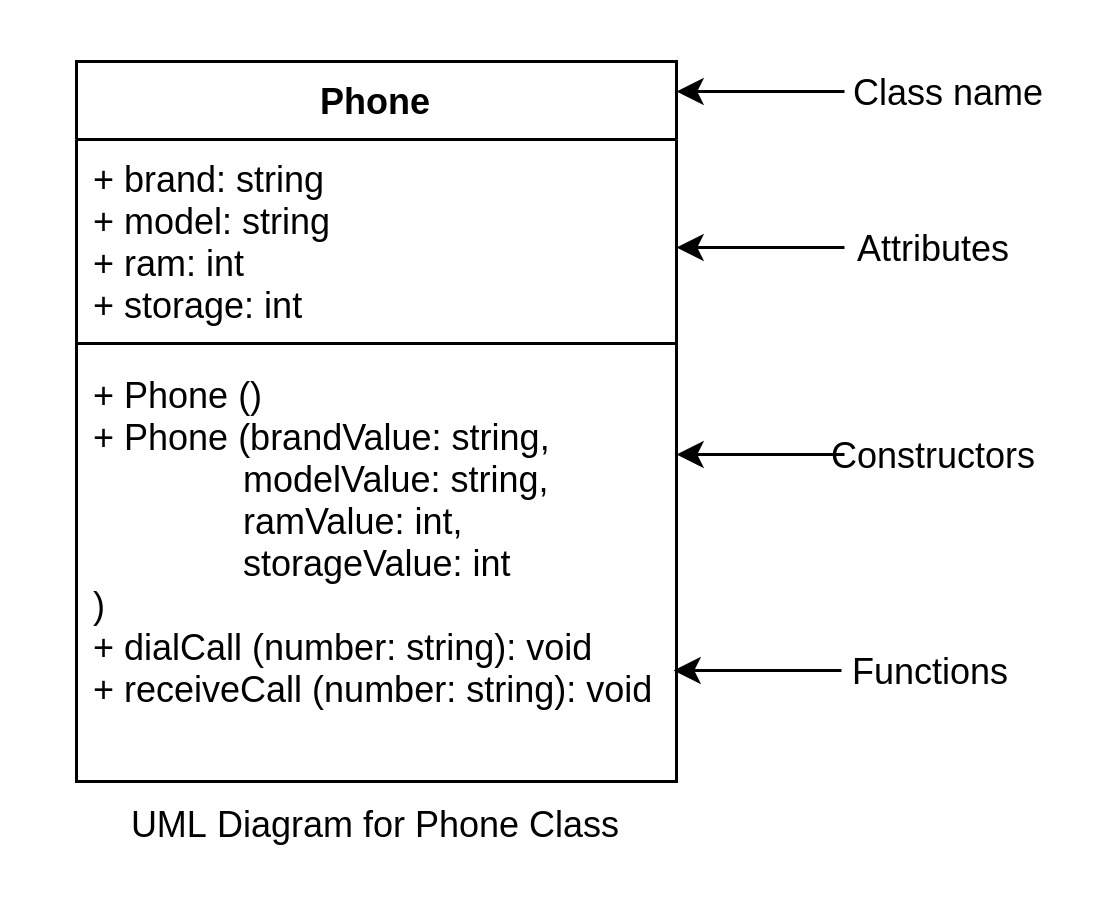
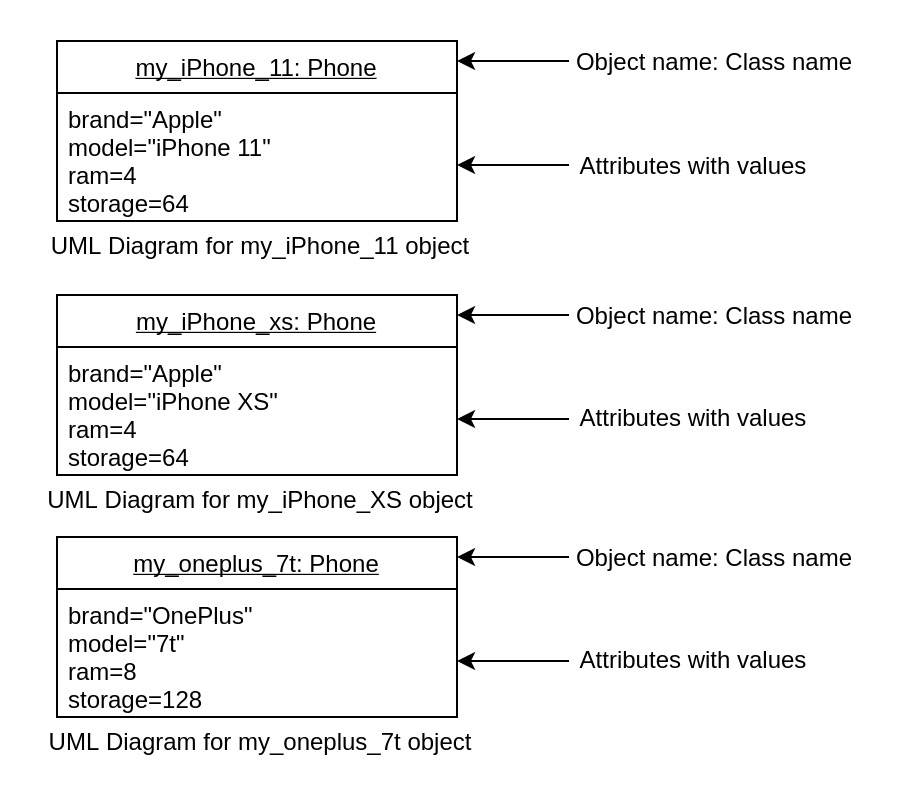
is equivalent to

Phone my\_iPhone\_11 = Phone();

Note that this will work only if:

* There are no constructors, or,
* There are one or more constructors and one of them takes no parameter. The one with no parameter acts as the default constructor.

Let's write some code with all the concepts learned till now. Doing this properly will help you retain these concepts and will make the assessment easier for you to complete.

* Create a class for the below UML Diagram.  
  
* Keep the default constructor with an empty code block.
* In the main method, create 3 objects of type Phone with the parameterized constructor:  
  
* Call dialCall and receiveCall after creating each of the objects.

#### Expected Output

Calling 9732130450 from Apple:iPhone 11

Receiving call from 9732130450 on Apple:iPhone 11

Calling 9732130450 from Apple:iPhone XS

Receiving call from 9732130450 on Apple:iPhone XS

Calling 9732130450 from OnePlus:7t

Receiving call from 9732130450 on OnePlus:7t

#include <bits/stdc++.h>

using namespace std;

class Phone {

public:

string brand;

string model;

int ram;

int storage;

Phone () {

//any initialization logic

}

Phone (string brandValue, string modelValue, int ramValue, int storageValue) {

brand = brandValue;

model = modelValue;

ram = ramValue;

storage = storageValue;

}

void dialCall(string number) {

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

}

void receiveCall(string number) {

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

}

};

int main() {

Phone my\_iPhone\_11 = Phone ("Apple", "iPhone 11", 4, 64);

Phone my\_iPhone\_XS = Phone ("Apple", "iPhone XS", 4, 64);

Phone my\_OnePlus\_7t = Phone ("OnePlus", "7t", 8, 128);

my\_iPhone\_11.dialCall("9732130450");

my\_iPhone\_11.receiveCall("9732130450");

my\_iPhone\_XS.dialCall("9732130450");

my\_iPhone\_XS.receiveCall("9732130450");

my\_OnePlus\_7t.dialCall("9732130450");

my\_OnePlus\_7t.receiveCall("9732130450");

return 0;

}

# Object-Oriented Programming - Basics in C++: Classes & Objects

## static class members

We've learned in previous sections that:

* We can create classes with some class members (attributes and functions).
* We can create objects which are instances of classes.
* We can also access the class members through these objects.

There are certain class members which we may want to access without creating an object of that class. Every object signifies a different entity and generally has a state (through its attributes) which might be different from each other.

##### Examples

Example 1

Let's say that you want to know the number of objects (phones) created from a particular class (Phone). To get that information we would not want to create another object of that class, right? Since every object has a different state and acts as different entities, we won't be able to know this information from any of the objects anyway as they don't know anything about one another.

We would want that information at the prototype level (in the class) from which the object is created. This is possible to do using static keyword. We will look at how to do it after all the examples.

Example 2

Let's say that we want to create a class named Calculator. It is supposed to have a lot of functions like:

* add(int firstNum, int secondNum),
* subtract(int firstNum, int secondNum),
* multiply(int firstNum, int secondNum),
* divide(int firstNum, int secondNum), etc.

One thing that we can do is create a class and add all these functions to the class and then create objects to use these functions.

The issue here is that every object takes up some memory. If we want to use these functions, we will have to create a separate object everywhere we would want to use it thereby taking up a lot of memory. This should not be required as these objects would not store any data that we might need. This is also possible to do using static keyword. We will look at how to do it after all the examples.

Example 3

In Example 1, we talked about the count variable in the Phone class. What if instead of accessing the count directly we want to expose a function which gives the number in a proper formatted like 10K, 1M, 1B, etc. This is also possible to do using static keyword.

Now, let's look at what actually is static keyword and how to use it.

Almost every Object-Oriented Language including C++ allows us to create static class members which are not part of the class instances (objects) and can be directly accessed from the class.

A static class attribute can be created by adding static keyword before the data type like this:

static data\_type variable\_name

Example:

class Phone {

static int count;

.

.

.

};

Note: The initial value of a static int variable is always 0

A static class function can be created in the same way by adding a static keyword before it. In this case, before the return type like this:

static return\_type function\_name(params) {

//Function logic

}

Example:

class Calculator {

static int add(int firstNumber, int secondNumber) {

return firstNumber + secondNumber;

}

.

.

.

};

**OOP Basics Introduction - Quiz 6**

Given a class Phone with a static attribute named count and certain non-static attributes. There is a static method as well. Which of these statements is true?

A. The static method will have access to count

B. The static method will have access to all attributes of Phone

C. The static members cannot be accessed through the class Phone

D. The non-static members can be directly accessed through the class Phone

Correct Answer is A.

Note that static class members are not tied to any object and so cannot access any of the non-static class members. Static class members can however access other static class members.

Example:

Here, count and getFormattedCount are both static members. Note that getFormattedCount is accessing the count attribute.

class Phone {

public:

static int count;

.

.

.

static string getFormattedCount() {

if (count/1000000000 > 0) {

return to\_string((double) count/1000000000) + "B";

}

else if (count/1000000 > 0) {

return to\_string((double) count/1000000) + "M";

}

else if (count/1000 > 0) {

return to\_string((double) count/1000) + "K";

} else {

return to\_string(count);

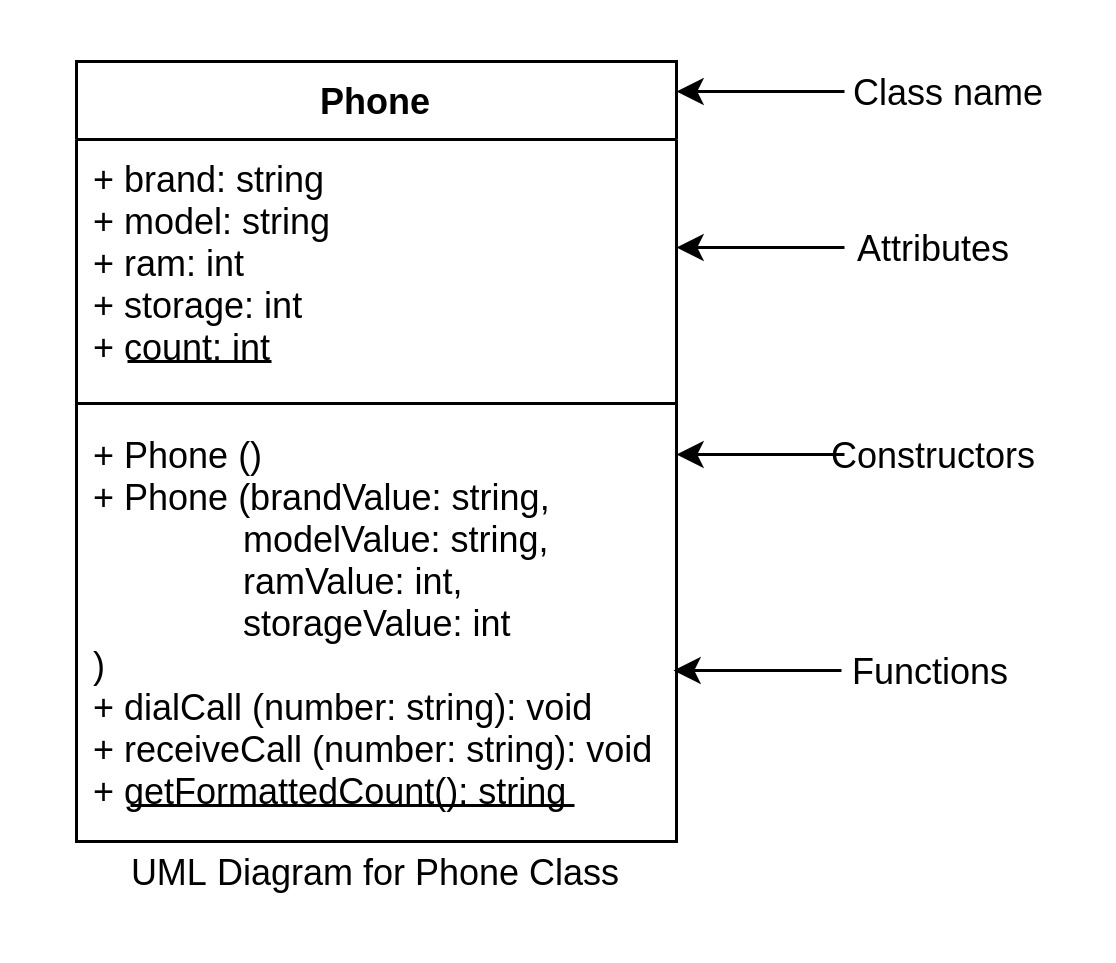
}

}

};

static class members are underlined in a UML Diagram.

Example:

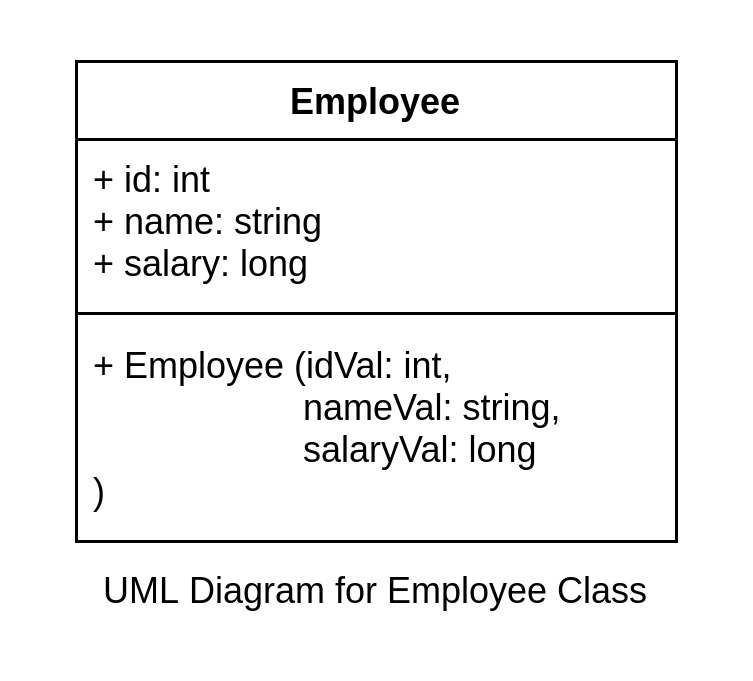


# Object-Oriented Programming - Basics in C++: Assessment - I

## Assessment - I

### Problem Statement:

Create a class Employee based on the following UML Diagram:



Do not modify the main method. Also use the same names and data types as used in the UML Diagram.

#include <bits/stdc++.h>

using namespace std;

class Employee{

public:

int id;

string name;

long salary;

Employee () {

//any initialization logic

}

Employee (int idVal, string nameVal, long salaryVal) {

id = idVal;

name = nameVal;

salary = salaryVal;

}

};

int main() {

// your code goes here

Employee hannibal\_lecter(1, "Hannibal Lecter", 10000000);

Employee norman\_bates(2, "Norman Bates", 9000000);

Employee darth\_vader(3, "Darth Vader", 8000000);

cout << hannibal\_lecter.id << " " << hannibal\_lecter.name << " " << hannibal\_lecter.salary << endl;

cout << norman\_bates.id << " " << norman\_bates.name << " " << norman\_bates.salary << endl;

cout << darth\_vader.id << " " << darth\_vader.name << " " << darth\_vader.salary << endl;

return 0;

}

**Object-Oriented Programming - Basics in C++: Encapsulation**

**Introduction**

Open your phone and open the calculator app. Try to visualize the calculator app on your phone in terms of an object of class CalculatorApp.

Which of these is least likely to be an attribute of class CalculatorApp?

A. name (Denoting the app name)

B. icon (Denoting the app icon)

C. theme (Denoting light/dark theme)

D. city (Denoting the city in which the app was developed)

Correct Answer is D.

C

D

Correct Answer is D.

Which of these would the app allow you to change as a user?

A. name (Denoting the app name)

B. icon (Denoting the app icon)

C. theme (Denoting light/dark theme)

D. developer (Denoting the company/person who developed the app)

A

B

C

D

Correct Answer is C.

Which of these would the app not allow you to see as a user?

A. name (Denoting the app name)

B. icon (Denoting the app icon)

C. theme (Denoting light/dark theme)

D. open\_count (Denoting the number of times you have opened the app)

A

B

C

D

Correct Answer is D.

If an app is supposed to allow/disallow you to access/modify a few of its attributes then there is a requirement for access control.

This is known as information hiding where one object or function is not allowed to access or modify an attribute of another object.

In this specific example, you are allowed to view the name of the app but neither you nor your phone is allowed to modify the name of the app.

The app might not allow you to view specific attributes as well such as the number of times you have opened the app. Your actions might indirectly modify that attribute but the app cannot allow you to modify that attribute directly.

In Object-Oriented Programming, this is done through Encapsulation.

**Encapsulation refers to the bundling of data with the methods that operate on that data. Here, we are bundling all the attributes (data) with the methods to create a class.**

By doing this we can restrict the direct access to some of an object's members by allowing read/write through other methods only.

How do we actually restrict access to attributes?

Access Specifier!

We've already used 'Access Specifiers' in a previous section.

Let's learn more about it in the next section.

**Object-Oriented Programming - Basics in C++: Encapsulation**

**Access Specifier/Modifier: public/private**

**Encapsulation - Quiz 2**

Which of these are class members?

A. Attributes and Functions

B. Only Attributes

C. Only Functions

D. None of the above

Correct Answer is A.

Let's look at what public and private means that we saw in the class definition template.

class ClassName {

private:

data\_type attribute\_name;

data\_type attribute\_name;

.

.

.

return\_type function\_name( args ) {

//Some logic

}

.

.

.

public:

data\_type attribute\_name;

data\_type attribute\_name;

.

.

.

return\_type function\_name( args ) {

//Some logic

}

.

.

.

};

public/private keywords used here are known as access specifiers or access control modifiers.

It is used to specify and control the visibility of a class member.

* public: Public class members are accessible everywhere.
* private: Private class members can be accessed only within that class.

Note that we can set any class member as private. It is not only for attributes. We use private to restrict access/modification of class members outside the class.

Let's make all the attributes of the Phone class as private.

class Phone {

private:

string brand;

string model;

int ram;

int storage;

public:

Phone () {

//any initialization logic

}

Phone (string brandValue, string modelValue, int ramValue, int storageValue) {

brand = brandValue;

model = modelValue;

ram = ramValue;

storage = storageValue;

}

void dialCall (string number) {

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

}

void receiveCall (string number) {

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

}

};

Now, none of the attributes of the Phone class are accessible to anyone outside the Phone class. We can still access these attributes inside the class. In the above example, we are using them in the constructor and the dialCall and receiveCall methods.

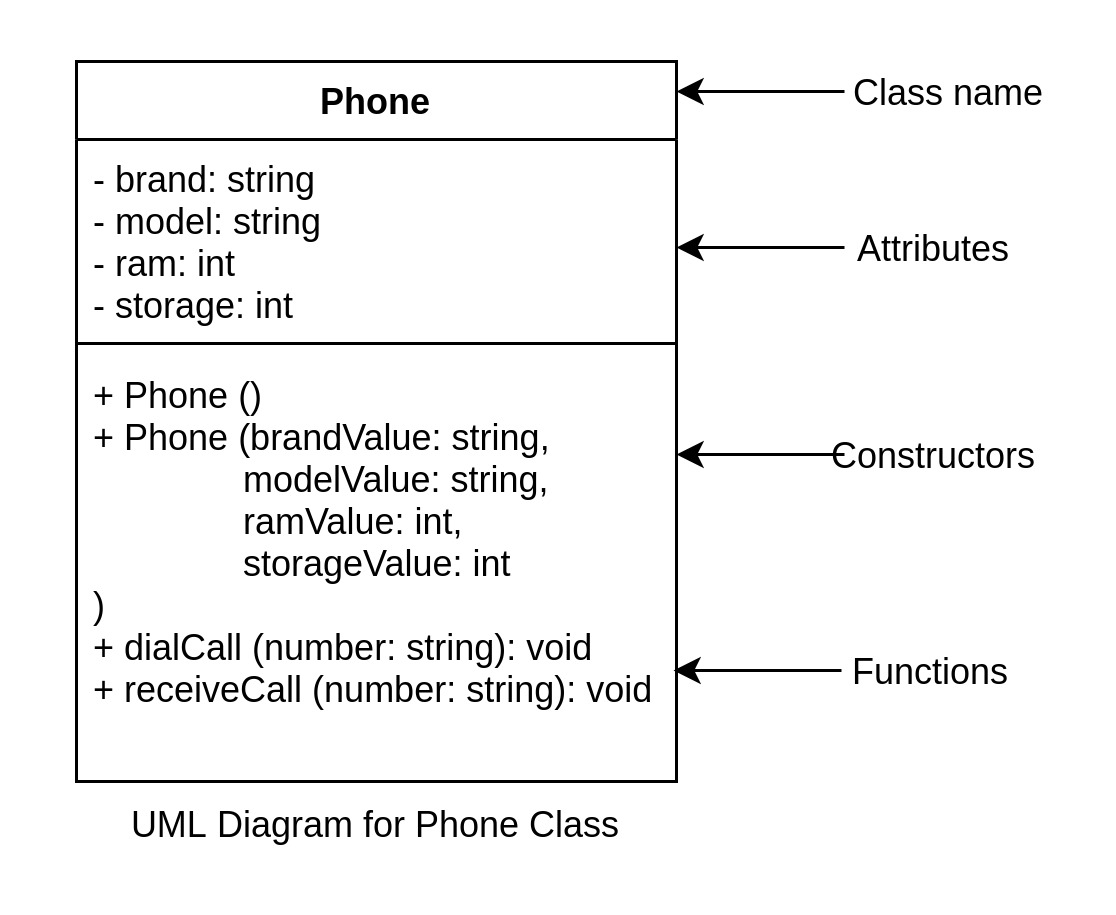
Now for an object my\_iPhone\_11 of class Phone, these things are invalid as the access is present only inside the class Phone:

* my\_iPhone\_11.brand = "Apple";
* cout << my\_iPhone\_11.brand;

Whereas these things are still valid:

* Phone my\_iPhone\_11 ("Apple", "iPhone 11", 4, 64);
* Phone my\_iPhone\_11 = Phone ("Apple", "iPhone 11", 4, 64);
* my\_iPhone\_11.dialCall ("9732130450");

The UML diagram of the above code would look like this:



Note that instead of +, the attributes are now prefixed with -.

Here, + is used to denote public class members whereas - is used to denote private class members.

In C++, all class members are by default private. If we do not specify public/private then the class member is treated as private.

The above class can also be written as:

class Phone {

string brand;

string model;

int ram;

int storage;

public:

Phone () {

//any initialization logic

}

Phone (string brandValue, string modelValue, int ramValue, int storageValue) {

brand = brandValue;

model = modelValue;

ram = ramValue;

storage = storageValue;

}

void dialCall (string number) {

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

}

void receiveCall (string number) {

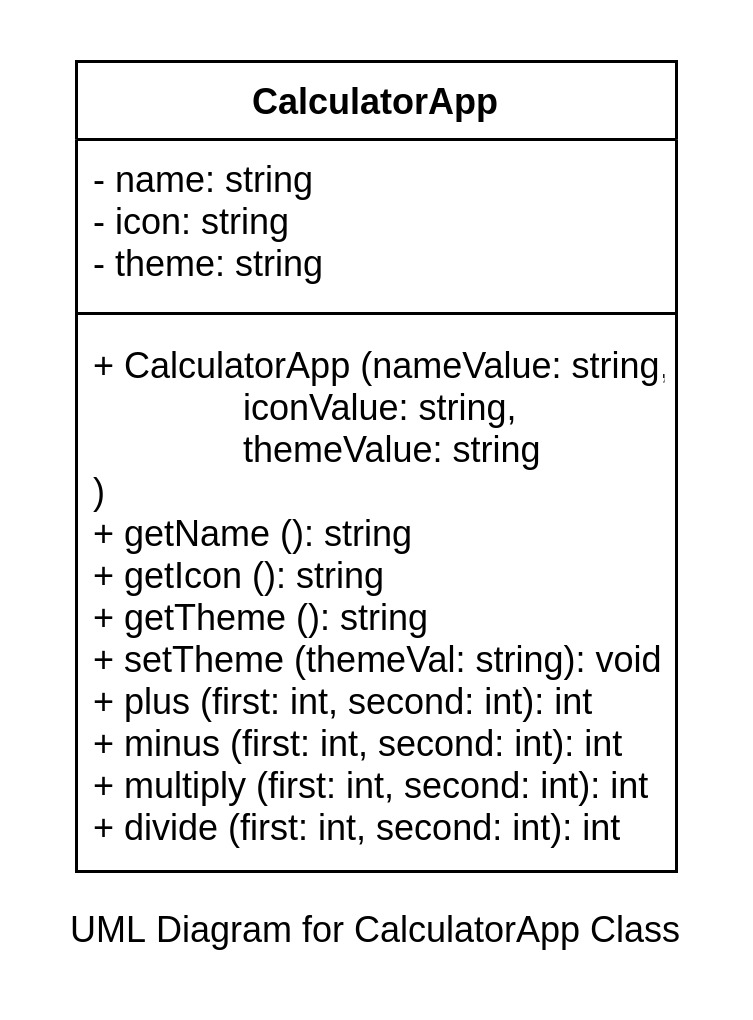
cout << "Receiving call from " << number << " on " << brand << ":" << model << "\n";

}

};

Here, all the attributes are treated as private and all members after the public keyword are treated as public.

Let's write some code.

* Create a class for the below UML class diagram.  
  
* getName, getIcon, and getTheme should return the name, icon, and theme respectively.
* setTheme should take themeVal as a parameter and should assign the passed value to theme. Look at how we do it in the constructor.  
  Example: After calling calculator.setTheme("Dark");  
  theme should be set to "Dark"
* plus, minus, multiply, divide should take two integers and return the result based on the operation (+, -, \*, /).

#include <bits/stdc++.h>

using namespace std;

class CalculatorApp{

private:

string name;

string icon;

string theme;

public:

CalculatorApp(){

}

CalculatorApp(string nameValue, string iconValue, string themeValue)

{

name = nameValue;

icon = iconValue;

theme = themeValue;

}

string getName(){

return name;

}

string getIcon(){

return icon;

}

string getTheme(){

return theme;

}

void setTheme(string themeVal)

{

theme = themeVal;

}

int plus(int first, int second){

return first+second;

}

int minus(int first, int second){

return first-second;

}

int multiply(int first, int second){

return first\*second;

}

int divide (int first, int second){

return first/second;

}

};

int main() {

// DO NOT MODIFY THE MAIN METHOD

CalculatorApp calculator ("Calculator", "/icon/calculator.jpg", "Light");

cout << "Name: " << calculator.getName() << endl;

cout << "Icon: " << calculator.getIcon() << endl;

cout << "Theme: " << calculator.getTheme() << endl;

calculator.setTheme("Dark");

cout << "Theme (after theme change): " << calculator.getTheme() << endl;

int firstNumber = 42, secondNumber = 8;

cout << "Plus: " << calculator.plus(firstNumber, secondNumber) << endl;

cout << "Minus: " << calculator.minus(firstNumber, secondNumber) << endl;

cout << "Multiply: " << calculator.multiply(firstNumber, secondNumber) << endl;

cout << "Divide: " << calculator.divide(firstNumber, secondNumber);

return 0;

}

**Output:**

Name: Calculator

Icon: /icon/calculator.jpg

Theme: Light

Theme (after theme change): Dark

Plus: 50

Minus: 34

Multiply: 336

Divide: 5

**Object-Oriented Programming - Basics in C++: Encapsulation**

**Pointer to Class**

**Encapsulation - Quiz 3**

For a class Phone, which of the options best describes the following code:

string brand;

Attribute/Property

Constructor

Object

Function

Correct Answer is A.

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}

ANSWER IS C.

As we've previously learned that we can access different properties of an object like this:

object.property

This is known as the dot member selection operator (or simply Dot Operator). It is used to access a class member through the object.

##### Example

class Phone {

public:

string brand;

string model;

int ram;

int storage;

Phone (string brandValue, string modelValue, int ramValue, int storageValue) {

brand = brandValue;

model = modelValue;

ram = ramValue;

storage = storageValue;

}

};

int main () {

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

//This will print the brand property of the object phone\_var.

cout << phone\_var.brand;

}

**Encapsulation - Quiz 4**

Given an int variable:

int var = 5;

How do you create a pointer to the int variable 'var'?

A.

int ptr = var;

B.

int \*ptr = var;

C.

int \*ptr = \*var;

D.

int \*ptr = &var;

A

B

C

D

Correct Answer is D.

Just like variables, objects are also stored in memory and take up some space. Since they are stored in memory, they will also have an address at which they are stored.

To get the address of a variable, we use a pointer. Similarly, we can create pointers to class objects as well.

##### Example

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

Phone \*ptr = &phone\_var;

Here ptr is a pointer variable of type Phone\* and points to the address of phone\_var.

Similar to how we can access class members of an object through the dot member selection operator, we can access them through the arrow member selection operator if we've a pointer to the object.

ClassName object;

ClassName \*obj\_ptr = &object;

cout << obj\_ptr->attribute\_name

##### Example

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

cout << phone\_var.brand << endl; //This will print Apple

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

cout << phone\_ptr->brand << endl; //This will print Apple

Arrow member selection operator (or simply Arrow Operator) is used to access a class member through a pointer to an object.

Note that arrow operators can be used on a class pointer but not on pointers of primitive data types (int, float, etc).

##### Example (Note that this is not valid)

int var = 5;

int \*ptr = &var;

cout << ptr->something; //This is not valid

Let's write some code. Modify the main method to create a pointer variable to object phone and print the ram using the arrow operator. Do not use the dot operator.

#### Expected Output

4

#include <bits/stdc++.h>

using namespace std;

class Phone {

public:

string brand;

string model;

int ram;

int storage;

Phone (string brandValue, string modelValue, int ramValue, int storageValue) {

brand = brandValue;

model = modelValue;

ram = ramValue;

storage = storageValue;

}

};

int main() {

//Created phone object

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

//Create a pointer variable to object phone

Phone \*p = &phone;

//Print the ram using the arrow operator. Do not use the dot operator.

cout << p->ram;

return 0;

}

**Object-Oriented Programming - Basics in C++: Encapsulation**

**this pointer**

For a class phone, which of the options best describes the following code:

Phone (string brandValue, string modelValue, int ramValue, int storageValue) {

brand = brandValue;

model = modelValue;

ram = ramValue;

storage = storageValue;

}

Attribute/Property

Constructor

Object

Class

Correct Answer is B.

For a class Phone, which of the options best describes the following code:

string brand;

Attribute/Property

Constructor

Object

Function

Correct Answer is A.

For a class phone, which of the options best describes the following code:

void setBrand(string brandVal) {

brand = brandVal;

}

Attribute/Property

Constructor

Object

Function

Correct Answer is D.

As you must have noticed that while passing a parameter to a constructor or any other class function, we have been using a parameter name that is different from the actual property name.

##### Examples:

Phone (string brandValue, string modelValue, int ramValue, int storageValue) {

brand = brandValue;

model = modelValue;

ram = ramValue;

storage = storageValue;

}

void setBrand(string brandValue) {

brand = brandVal;

}

We have been using variable names like brandValue, modelValue, ramValue, storageValue, etc instead of brand, model, ram, storage. We are doing to avoid something like this:

//Note that this will not work

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

brand = brand;

model = model;

ram = ram;

storage = storage;

}

OR

//Note that this will not work

void setBrand(string brand) {

brand = brand;

}

**Encapsulation - Quiz 6**

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.

##### Example of "this" pointer

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

this->brand = brand;

this->model = model;

this->ram = ram;

this->storage = storage;

}

void setBrand(string brand) {

this->brand = brand;

}

Here, "this" pointer refers to a pointer to the actual object on which the function is being called. In the case of a constructor, it is the pointer to the object that will be created by the constructor.

In the above example where we're doing:

this->brand = brand;

* this->brand denotes the brand attribute of the object
* brand denotes the function parameter

this pointer can be used to reference the object only inside its class. We can use this along with an arrow operator anywhere inside the class to access or modify any attribute in the object or to call a method internally.

##### Examples:

void setBrand(string brand) {

this->brand = brand;

}

void getBrand(string brand) {

return this->brand;

}

void incrementValue() {

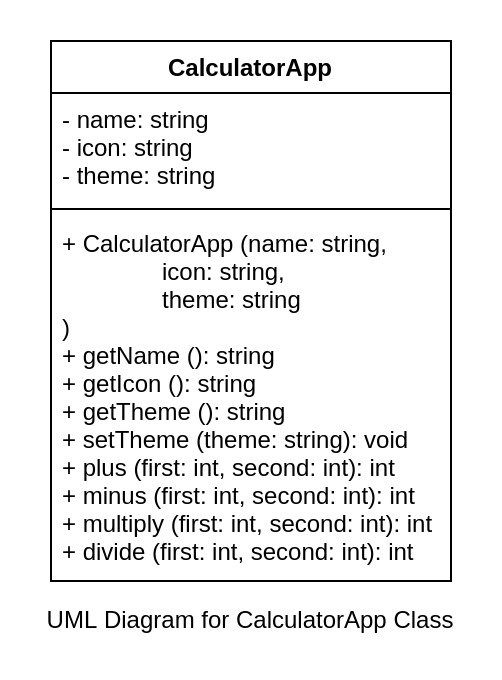
this->setValue(this->value + 1);

}

Note that "this->" is mandatory only when we've another variable with the same name as a class member, i.e., when there is ambiguity between a variable name and a class member. This is why we didn't have to use this-> when we were using a different parameter name.

In the above code we can remove this-> from inside the incrementValue method as there are no ambiguity there.

Let's write some code. We've already written a similar code in a previous section. The only difference being that now you need to use this pointer in the constructor, set methods, and get methods.

* Create a class for the below UML class diagram.  
  
* getName, getIcon, and getTheme should return the name, icon, and theme respectively.
* setTheme should take theme as a parameter and should assign the passed value to this->theme.  
  Example: If we call calculator.setTheme("Dark");  
  theme should be set to "Dark" in the method.
* plus, minus, multiply, divide should take two integers and return the result based on the operation (+, -, \*, /).

#include <bits/stdc++.h>

using namespace std;

class CalculatorApp{

private:

string name;

string icon;

string theme;

public:

CalculatorApp(){}

CalculatorApp(string name, string icon, string theme){

this->name=name;

this->icon=icon;

this->theme=theme;

}

string getName(){

return name;

}

string getIcon(){

return icon;

}

string getTheme(){

return theme;

}

void setTheme(string theme){

this->theme=theme;

}

int plus(int first, int second){

return first+second;

}

int minus(int first, int second){

return first-second;

}

int multiply(int first, int second){

return first\*second;

}

int divide(int first, int second){

return first/second;

}

};

int main() {

// DO NOT MODIFY THE MAIN METHOD

CalculatorApp calculator ("Calculator", "/icon/calculator.jpg", "Light");

cout << "Name: " << calculator.getName() << endl;

cout << "Icon: " << calculator.getIcon() << endl;

cout << "Theme: " << calculator.getTheme() << endl;

calculator.setTheme("Dark");

cout << "Theme (after theme change): " << calculator.getTheme() << endl;

int firstNumber = 42, secondNumber = 8;

cout << "Plus: " << calculator.plus(firstNumber, secondNumber) << endl;

cout << "Minus: " << calculator.minus(firstNumber, secondNumber) << endl;

cout << "Multiply: " << calculator.multiply(firstNumber, secondNumber) << endl;

cout << "Divide: " << calculator.divide(firstNumber, secondNumber);

return 0;

}

**Object-Oriented Programming - Basics in C++: Encapsulation**

**Getters and Setters**

Which of these would you use to restrict access to a class attribute?

public

private ANSWER

personal

local

Which of these can be made public/private?

A. Class Members (Attributes and Functions) ANSWER

B. Only Attributes

C. Only Functions

D. None of the above

If we want to allow someone to read/get the value of a particular attribute, say theme, but do not want it to be open for modification which of the following methods should we add in our class?

string getTheme() {

return this->theme;

}

void setTheme(string theme) {

this->theme = theme;

}

A. Only getTheme ANSWER

B. Only setTheme

C. Both getTheme and setTheme

D. None of the above

If we want to allow someone to read/get the value of a particular attribute, say theme, and also want it to be open for modification which of the following methods should we add in our class?

string getTheme() {

return this->theme;

}

void setTheme(string theme) {

this->theme = theme;

}

A. Only getTheme

B. Only setTheme

C. Both getTheme and setTheme ANSWER

D. None of the above

It is always a good idea to keep all the class attributes as private. As we have seen in the previous section, we can expose an attribute, say theme, like this:

class CalculatorApp {

private:

string theme;

public:

CalculatorApp (string theme) {

this->theme = theme;

}

string getTheme() {

return this->theme;

}

void setTheme(string theme) {

this->theme = theme;

}

};

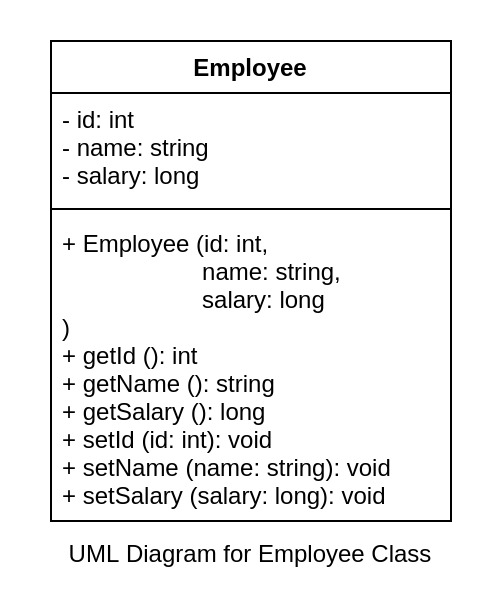
These methods are commonly known as getters and setters. They are widely used in commercial applications where object-oriented programming is used.

# Object-Oriented Programming - Basics in C++: Assessment - II

## Assessment - II

### Problem Statement:

Create a class Employee based on the following UML Diagram:



Do not modify the main method. Also use the same names and data types as used in the UML Diagram.

#include <bits/stdc++.h>

using namespace std;

class Employee{

private:

int id;

string name;

long salary;

public:

Employee(){}

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;

}

void setId(int id){

this->id = id;

}

void setName(string name){

this->name=name;

}

void setSalary(long salary){

this->salary=salary;

}

};

int main() {

// your code goes here

Employee hannibal\_lecter(1, "Hannibal Lecter", 10000000);

Employee norman\_bates(2, "Norman Bates", 9000000);

Employee darth\_vader(3, "Darth Vader", 8000000);

darth\_vader.setId(327);

darth\_vader.setName("Anakin Skywalker");

darth\_vader.setSalary(7500000);

cout << hannibal\_lecter.getId() << " " << hannibal\_lecter.getName() << " " << hannibal\_lecter.getSalary() << endl;

cout << norman\_bates.getId() << " " << norman\_bates.getName() << " " << norman\_bates.getSalary() << endl;

cout << darth\_vader.getId() << " " << darth\_vader.getName() << " " << darth\_vader.getSalary() << endl;

return 0;

}

**Output:**

1 Hannibal Lecter 10000000

2 Norman Bates 9000000

327 Anakin Skywalker 7500000