1. Classes and Objects

What is a Class?

A **class** is a blueprint for creating objects. It defines a set of properties (attributes) and methods (behaviors) that the objects created from the class will have.

Imagine a class as a recipe for baking a cake. The recipe (class) includes ingredients (attributes) and steps (methods) to bake the cake.

General Syntax:

```
class (keyword) className {
public: (access specier)
Class Members
}
```

Example:

```
class Recipe {
        public:
                // Attributes
                string flavor;
                int layer;

                // Method
                void BakeIt(){
                        cout << "Baking the " << flavor << " of " << layer << " layers. " << endl;
                }
};
```

Figure 1 Recipe is the Class

NOTE: A class all the information that is required to create an object. Or you can say that an object will be exactly similar to its class.

What is an Object?

An **object** is an instance of a class. Using the recipe analogy, the actual cake you bake using the recipe is an object. You can create multiple cakes (objects) using the same recipe (class).

Now we will create an object of class Recipe we created earlier.

The objects are created and used inside the main() function. While classes are created before it

Example:

```
int main(){
        Recipe myCake; // Object
        myCake.flavor = "Chocolate Cake";
        myCake.layer = 3;

        myCake.BakeIt();

        return 0;
}
```

Figure 2 Cakes are the recipes that are created from the class

2. Class, Instance Variables, and Methods

Class Definition

A class is defined using the class keyword followed by the class name and a pair of curly braces {} containing the class members.

Class members contains basically the details inside the class and will tell about the structure of the object that will be created with that class.

It is of two types.

Figure 3 Flavor and layer are instance variables, Baking is method

Instance Variables

Instance variables **(attributes)** are the properties of an object. They are defined within the class.

For example: In our Recipe class example, **flavor** and **layer** are the instance variables representing two attributes of the cakes (objects) that will be created.

Methods

Methods **(functions)** are the behaviors of an object. They define what actions the object can perform.

For example: In Recipe class, **BakeIt()** is the method associated with the class.

NOTE: Methods are basically C++ functions bound to a class.

OOP with C++

Here is another example:

```
class Car {
public:
   string brand;
   int year;

   void drive() {
      cout << "Driving a " << brand << " from " << year << "." << endl;
   }
};

int main() {
   Car myCar;
   myCar.brand = "Toyota";
   myCar.year = 2020;

   myCar.drive();  // Output: Driving a Toyota from 2020.

   return 0;
}
```

3. Access Specifiers in C++

Access specifiers in C++ are keywords used to set the accessibility or visibility of class members (variables and methods). They control how the data in a class is accessed and manipulated. The three main access specifiers in C++ are:

1. **Public**
2. **Private**
3. **Protected**

Each of these access specifiers determines how the members of the class can be accessed from other parts of the program.

ACCESS	PUBLIC	PROTECTED	PRIVATE
SAME CLASS	Yes	Yes	Yes
DERIVED CLASS	Yes	Yes	No
OUTSIDE CLASS	Yes	No	No

Let's see them one by one in details.

OOP with C++

1. Public

Consider a library. The books on the shelves are public—they're accessible to anyone who enters the library. You can browse, read, and borrow the books without needing special permission. In programming, **public** members of a class are like those books in the library. They can be accessed and used by any part of the program. If a class has a public method, any other part of the code can call that method.

Public members are accessible from anywhere in the program. If a member of a class is declared as public, it can be accessed directly using the object of the class.

Example: See the Recipe and Car Example shared above.

Analogy: Think of a public park, public library etc.

2. Private

In a company, your personal desk drawer might be private. It's where you keep personal items like your wallet, keys, or confidential documents. Only you have the key, and no one else can access it without your permission. In programming, **private** members of a class are like that locked drawer.

Private members are accessible only within the class itself. They cannot be accessed directly from outside the class, ensuring data encapsulation and protecting the integrity of the data.

NOTE: Private method is linked to encapsulation. (How? We will discuss).

Example:

```
class BankAccount {
private:
   double balance;

public:
   void deposit(double amount) {
      if (amount > 0) {
         balance += amount;
      }
   }

   double getBalance() {
      return balance;
   }
};
```

OOP with C++

```
int main() {
    BankAccount account;
    account.deposit(1000);
    cout << "Balance: " << account.getBalance() << endl;  // Output: Balance: 1000

    // account.balance = 2000;  // Error: 'balance' is a private member of 'BankAccount'

    return 0;
}
```

We can see that we cannot directly access the private variable (balance), because it is encapsulated inside the class only. However, we used to separate Public Members to get access to the balance and interact with it.

Analogy: Think of a private safe in your home.

A **private** safe is only accessible by you (*and maybe a few trusted individuals, member functions!*). No one else can open it or see what's inside. You control what goes in and out, and you decide who, if anyone, gets access.

3. Protected

Imagine you work in a company, and your manager has certain information that only their team members are allowed to see. This information is protected within the team—it's not available to the entire company, but team members can access it because they work closely with the manager.

Protected members are similar to private members but can also be accessed by derived classes (classes that inherit from the parent class). Protected members are used primarily in inheritance scenarios.

NOTE: We will learn about inheritance later. So far, only discussing it for a general idea.

4. Encapsulation

Concept

Encapsulation is the concept of bundling the data (instance variables) and methods that operate on the data into a single unit (class), and restricting access to some of the object's components.

Access Specifiers

- **Public**: Members declared as public are accessible from outside the class.
- **Private**: Members declared as private are accessible only within the class.

OOP with C++

Example: Look at the Bank Account Example again.

Encapsulation ensures bundling of both public and private data and members in a class.

Here, we bundled the private balance and used public methods (deposit and getBalance) to get and modify the private data.

Question is: Why Encapsulation why? ☹

Encapsulation is a way to restrict the direct access to some components of an object, so users cannot access state values for all of the variables of a particular object. Encapsulation is a programming concept that restricts direct access to certain components of an object, such as data members and methods. It's a way to hide internal details and only expose necessary functionality.

Can be used:

- Prevent unwanted access by clients
- Encapsulation is applicable in real-world scenarios, such as in marketing and finance, where security and restricted data access are important.
- Encapsulation ensures better control of your data, because you (or others) can change one part of the code without affecting other parts.

Note: By default, all members of a class are private if you don't specify an access specifier.

5. Get and Set Functions

Purpose

Get and set functions (accessors and mutators) are used to control access to the instance variables. They provide a way to read (get) and modify (set) the values of private variables.

To access a private attribute, use public "get" and "set" methods.

```
class Person {
private:
  string name;
  int age;

public:
  void setName(string n) {
    name = n;
  }

  string getName() {
```

OOP with C++

```
    return name;
  }

  void setAge(int a) {
    age = a;
  }

  int getAge() {
    return age;
  }
};

int main() {
  Person person;
  person.setName("Wasiq");
  person.setAge(37);

  cout << "Name: " << person.getName() << ", Age: " << person.getAge() << endl;

  return 0;
}
```

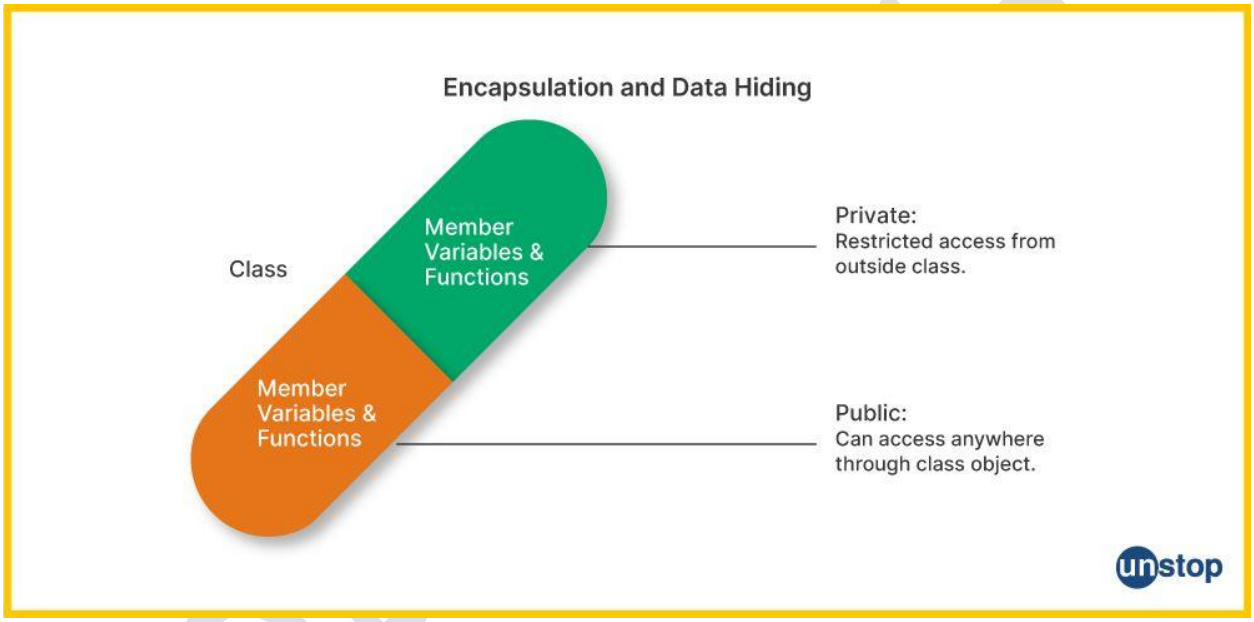
Another example:

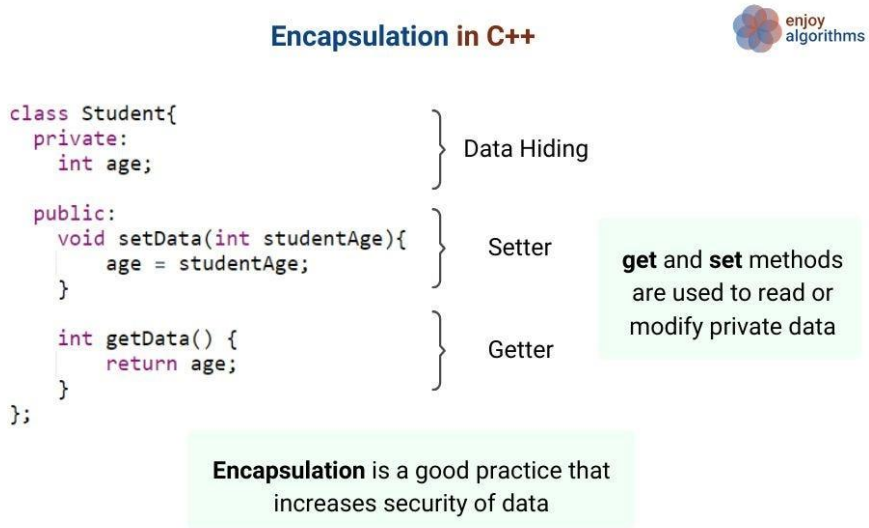
```
class Employee {
 private:
  // Private attribute
  int salary;

 public:
  // Setter
  void setSalary(int s) {
    salary = s;
  }
  // Getter
  int getSalary() {
    return salary;
  }
};

int main() {
 Employee myObj;
 myObj.setSalary(50000);
 cout << myObj.getSalary();
 return 0;
}
```

OOP with C++

Example explained

- The salary attribute is private, which have restricted access.
- The public setSalary() method takes a parameter (s) and assigns it to the salary attribute (salary = s).
- The public getSalary() method returns the value of the private salary attribute.
- Inside main(), we create an object of the Employee class. Now we can use the setSalary() method to set the value of the private attribute to 50000. Then we call the getSalary() method on the object to return the value.





CLASS ONE PRACTICE QUESTIONS:

Reasoning-Type Questions

1. Explain the difference between public and private access specifiers with a real-world example.
2. Why is encapsulation important in Object-Oriented Programming? Provide an example to support your answer.
3. What is the purpose of get and set functions in a class? How do they help in maintaining data integrity?
4. If you want a class member to be accessible by derived classes but not by unrelated classes, which access specifier would you use? Explain with an example.
5. Consider a class **Book** with a private attribute **price**. How would you allow other parts of the program to read and modify the **price**?
6. Why should sensitive data members of a class generally be declared as private? Provide an example scenario.
7. What would happen if all members of a class were declared public? Discuss the potential risks.
8. Can a private member of a class be accessed directly outside of the class? If not, how can it be accessed?
9. In what scenario would you prefer using protected access specifier over private? Provide a practical example.
10. How does encapsulation contribute to the concept of modularity in software design?

Programming Questions

1. **Basic Class Creation:**
   - Create a class Student with the following private attributes: name, age, and grade. Implement public methods to set and get each of these attributes. Write a main function to create an object of Student, set the attributes, and print them.
2. **Encapsulation and Access Control:**
   - Create a class Account with a private attribute balance. Implement methods to deposit and withdraw money, ensuring that the balance cannot become negative. Also, create a method to check the current balance. Write a main function to simulate some transactions.
3. **Public vs Private:**
   - Modify the Car class from previous examples by making the attributes brand and year private. Add appropriate get and set methods to access and modify these attributes. Write a main function to demonstrate the use of these methods.
4. **Protected Members:**
   - Create a base class Animal with a protected attribute species. Create a derived class Dog that sets the species attribute through a constructor and prints it. Write a main function to create an object of the Dog class and display its species.
5. **Get and Set Functions:**
   - Create a class Rectangle with private attributes length and width. Implement get and set methods for both attributes, ensuring that only positive values can be assigned. Create a

OOP with C++

method to calculate the area of the rectangle. Write a main function to create a rectangle object, set its dimensions, and print its area.

6. **Constructor Usage:**
   - Create a class Book with private attributes title, author, and price. Implement a constructor to initialize these attributes. Add get methods for each attribute. Write a main function to create a Book object using the constructor and display its details.
7. **Inheritance and Access Control:**
   - Create a base class Employee with private attributes name and salary, and protected attribute position. Create a derived class Manager that sets the position and adds a method to display the manager's details. Write a main function to demonstrate inheritance and access control.
8. **Validating Inputs:**
   - Create a class Temperature with a private attribute celsius. Implement a method to set the temperature in Celsius, ensuring that the value is within a realistic range (-273.15 to 1000). Add a method to convert the temperature to Fahrenheit. Write a main function to set a temperature and display it in both Celsius and Fahrenheit.
9. **Multiple Objects:**
   - Create a class Point representing a point in 2D space with private attributes x and y. Implement get and set methods for these attributes. Write a main function to create multiple Point objects and demonstrate setting and getting their coordinates.
10. **Default Values:**
    - Create a class Person with private attributes name and age. Implement a constructor that sets default values for these attributes if no values are provided. Write a main function to create objects of Person using both default and custom values, then display the details.

OOP with C++