OOPS

Encapsulation

Encapsulation is one of the key features of object-oriented programming. It involves the ***bundling*** of data ***members*** and ***functions*** inside a single class.

Bundling similar data members and functions inside a class together also helps in ***data hiding***.

In general, encapsulation is a process of ***wrapping similar code*** in one place.

In C++, we can bundle data members and functions that operate together inside a single class. For example,

```
#include <bits/stdc++.h>
using namespace std;
class Cuboid {
  public:
   int length, breadth, height;
   // Constructor to initialize Values
   Cuboid(int l, int b, int h) : length(l), breadth(b), height(h) {}

   // Function to Calculate Volume
   int getVolume() { return length * breadth * height; }
};

int main() {
   Cuboid c(1, 2, 3);
   cout << "Volume is: " << c.getVolume();
}
```

In the above example, we are calculating the volume of a cuboid.

To calculate the volume, we need three variables: ***length***, ***height*** and ***breadth*** and a function: ***getVolume()***. Hence, we ***bundled*** these variables and function inside a single class named Cuboid.

Here, the variables and functions can be accessed from other classes as well. Hence, this is ***not data hiding***.

This is only ***encapsulation***. We are just keeping similar codes together.

> **Note:** People often consider encapsulation as data hiding, but that's not entirely true.
>
> Encapsulation refers to the bundling of related fields and methods together. This can be used to achieve data hiding. ***Encapsulation in itself is not data hiding.***

Why Encapsulation?

In C++, encapsulation helps us keep ***related data and functions together***, which makes our code cleaner and easy to read.
It helps to control the modification of our data members.

Consider a situation where we want the length field in a class to be non-negative. Here we can make the length variable private and apply the logic inside the method ***setLength()***. For example,

```
#include <bits/stdc++.h>
using namespace std;
class Cuboid {
  private:
   int length;

  public:
   void setlength(int len) {
      if (len >= 0) length = len;
   }
};

int main() {
```

```
   Cuboid c;
   c.setlength(7);
   return 0;
}
```

The getter and setter functions provide read-only or write-only access to our class members. For example,

```
getLength()  // provides read-only access
setLength()  // provides write-only access
```

It helps to decouple components of a system. For example, we can encapsulate code into multiple bundles. These decoupled components (bundles) can be developed, tested, and debugged independently and concurrently. And any changes in a particular component do not have any effect on other components.

We can also ***achieve data hiding using encapsulation***. In Example 1, if we change the length, breadth and height variables into ***private*** or ***protected***, then the access to these variables is restricted. And, they are kept hidden from outer classes. This is called ***data hiding***.

Data Hiding

Data hiding is a way of restricting the access of our data members by hiding the implementation details. Encapsulation also provides a way for data hiding.

We can use access modifiers to achieve data hiding in C++. For example,

```
#include <bits/stdc++.h>
using namespace std;
class Cuboid {
   int length, breadth, height;

  public:
   // Setter functions
   void setLength(int l){length = l;}
   void setBreadth(int b){breadth = b;}
   void setHeight(int h){height = h;}

   // Getter Functions
   int getLength() { return length; }
   int getBreadth() { return breadth; }
   int getHeight() { return height; }

   // Function to Calculate Volume
   int getVolume() { return length * breadth * height; }
};

int main() {
   Cuboid c;
   c.setLength(1);
   c.setBreadth(2);
   c.setHeight(3);

   cout << "Length: " << c.getLength() << "\n";
   cout << "Breadth: " << c.getBreadth() << "\n";
   cout << "Height: " << c.getHeight() << "\n";
   cout << "Volume is: " << c.getVolume();
}
```

Output:

```
Length: 1
Breadth: 2
Height: 3
Volume is: 6
```

Here, we have made the length, breadth and height variables private.

This means that these variables cannot be directly accessed outside of the Cuboid class.

To access these private variables, we have used public functions ***setLength(), getLength(), setBreadth(), getBreadth(), setHeight(), getHeight()***. These are called ***getter and setter functions***.

Making the variables private allowed us to ***restrict unauthorized access from outside the class***. This is ***data hiding***.

If we try to access the variables from the ***main()*** class, we will get an error.

```
// error: c.length is inaccessible
c.length = 8;

// error: c.breadth is inaccessible
c.breadth = 6;
```