**C++ Encapsulation**

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,

class Rectangle {

public:

int length;

int breadth;

int getArea() {

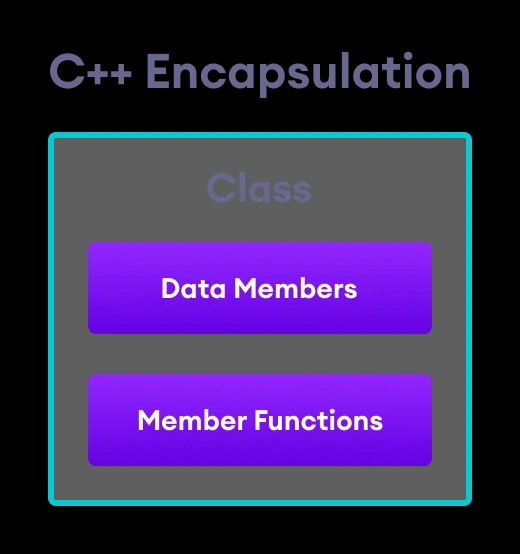
return length \* breadth;

}

};

In the above program, the function getArea() calculates the area of a rectangle. To calculate the area, it needs length and breadth.

Hence, the data members (length and breadth) and the function getArea() are kept together in the Rectangle class.



**C++ Encapsulation**

// Program to calculate the area of a rectangle

#include <iostream>

using namespace std;

class Rectangle {

public:

// Variables required for area calculation

int length;

int breadth;

// Constructor to initialize variables

Rectangle(int len, int brth) : length(len), breadth(brth) {}

// Function to calculate area

int getArea() {

return length \* breadth;

}

};

int main() {

// Create object of Rectangle class

Rectangle rect(8, 6);

// Call getArea() function

cout << "Area = " << rect.getArea();

return 0;

}

**Output**

Area = 48

In the above example, we are calculating the area of a rectangle.

To calculate an area, we need two variables: length and breadth and a function: getArea(). Hence, we bundled these variables and function inside a single class named Rectangle.

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 setAge(). For example,  
      
    class Rectangle {

private:

int age;

public:

void setLength(int len) {

if (len >= 0)

length = len;

}

};

* + 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 and breadth variables into private or protected, then the access to these fields 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,

**Example 2: C++ Data Hiding Using the private Specifier**

#include <iostream>

using namespace std;

class Rectangle {

private:

// Variables required for area calculation

int length;

int breadth;

public:

// Setter function for length

void setLength(int len) {

length = len;

}

// Setter function for breadth

void setBreadth(int brth) {

breadth = brth;

}

// Getter function for length

int getLength() {

return length;

}

// Getter function for breadth

int getBreadth() {

return breadth;

}

// Function to calculate area

int getArea() {

return length \* breadth;

}

};

int main() {

// Create object of Rectangle class

Rectangle rectangle1;

// Initialize length using Setter function

rectangle1.setLength(8);

// Initialize breadth using Setter function

rectangle1.setBreadth(6);

// Access length using Getter function

cout << "Length = " << rectangle1.getLength() << endl;

// Access breadth using Getter function

cout << "Breadth = " << rectangle1.getBreadth() << endl;

// Call getArea() function

cout << "Area = " << rectangle1.getArea();

return 0;

}

**Output**

Length = 8

Breadth = 6

Area = 48

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

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

To access these private variables, we have used public functions setLength(), getLength(), setBreadth(), and getBreadth(). 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: rectangle1.length is inaccessible

rectangle1.length = 8;

// error: rectangle1.breadth is inaccessible

rectangle1.length = 6;