# Encapsulation vs Abstraction

## Encapsulation vs. Abstraction in Object-Oriented Programming (OOP)

Both **Encapsulation** and **Abstraction** are fundamental principles in Object-Oriented Programming (OOP), but they serve different purposes. Let's break down both concepts in simple terms:

---

## Encapsulation

Encapsulation is the process of wrapping data (variables) and methods (functions) that operate on the data into a single unit or class. It restricts direct access to some of an object's components, which is useful for hiding the internal representation or state of the object from the outside world.

**Key Points:**

- **Data Hiding**: Encapsulation allows data hiding. You can hide the internal state of an object and require all interactions to be performed through an object's methods.
- **Control Access**: By using access modifiers (like `private`, `protected`, `public` in C++), you control who can access the data.
- **Benefits**: It increases security and prevents the accidental modification of the data.

**Example (Encapsulation in C++):**

cpp

```cpp
#include <iostream>
using namespace std;

class Car {
private:
    int speed; // Private variable, not accessible directly

public:
    // Setter method to modify speed
    void setSpeed(int s) {
        if (s > 0) speed = s; // Simple validation
    }

    // Getter method to access speed
    int getSpeed() {
        return speed;
    }
};

int main() {
    Car car;
    car.setSpeed(150);  // Set speed using setter
    cout << "Car speed is: " << car.getSpeed() << " km/h" << endl; // Get speed using
getter
    return 0;
}
```

**Explanation:**

- The `speed` variable is **encapsulated** within the `Car` class.
- It is marked as `private`, meaning it can't be accessed directly outside the class.
- Instead, the setter (`setSpeed()`) and getter (`getSpeed()`) methods control how it is accessed and modified.

---

## Abstraction

Abstraction refers to the concept of showing only the necessary details and hiding the complex implementation. It helps in reducing programming complexity and effort by allowing the programmer to work at a higher level without needing to understand the inner workings.

**Key Points:**

- **Focus on What, Not How**: Abstraction is about hiding the implementation details and exposing only the essential features.
- **Reduce Complexity**: It simplifies complex systems by breaking them into smaller, understandable parts.
- **Achieved Through**: Abstract classes and interfaces in languages like Java, or simply through methods in C++.

**Example (Abstraction in C++):**

cpp

```cpp
#include <iostream>
using namespace std;

// Abstract class
class Shape {
public:
    // Pure virtual function (abstract method)
    virtual void draw() = 0;
};

class Circle : public Shape {
public:
    void draw() override {
        cout << "Drawing a Circle" << endl;
    }
};

class Rectangle : public Shape {
public:
    void draw() override {
        cout << "Drawing a Rectangle" << endl;
    }
};

int main() {
    Shape* shape1 = new Circle();
    Shape* shape2 = new Rectangle();

    shape1->draw();  // Draw a Circle
    shape2->draw();  // Draw a Rectangle

    delete shape1;
    delete shape2;
```

```
    return 0;
}
```

**Explanation:**

- The `Shape` class defines an abstract concept (the idea that shapes can be drawn) without specifying how to draw them.
- The `Circle` and `Rectangle` classes provide specific implementations of the `draw()` method.
- The abstraction hides the complex implementation details of drawing each shape and focuses on the essential feature: "What needs to be done?" (i.e., drawing).

---

## Main Differences:

| Feature | Encapsulation | Abstraction |
|---|---|---|
| **Purpose** | Bundling data and methods that operate on it, hiding internal state | Hiding complex details, focusing on high-level operations |
| **Access Control** | Uses access specifiers (`private`, `public`, `protected`) | Focuses on exposing only necessary interfaces (e.g., abstract classes, methods) |
| **Focus** | Protects object integrity by controlling access | Reduces complexity by showing only relevant details |
| **Example** | Using getters/setters to hide and control access to variables | Abstract classes and interfaces that hide the implementation details |
| **Visibility** | Hides the "how" of data manipulation | Hides the "how" of an operation or functionality |

---

## Conclusion:

- **Encapsulation** is about hiding the data to protect it from unauthorized access and ensuring the integrity of the object's state.
- **Abstraction** is about hiding the complexity by providing a simpler interface that focuses on "what" the object or system does rather than "how" it does it.

In simple terms, encapsulation is like putting sensitive documents in a safe (hiding them but allowing controlled access), and abstraction is like presenting only the summary of those documents instead of the entire detailed report.