Setters, getters and constructors

Encapsulation & Data Hiding

- Encapsulation involves bundling the data (attributes or properties) and methods (functions) that operate on that data into a single unit, called a class. This allows for better organization of code and improves the security and integrity of the data.
- Data hiding is a key aspect of encapsulation, where the internal representation of an object is hidden from the outside world. This prevents external code from directly accessing or modifying the internal state of an object, thus ensuring data integrity and security.

- In this example, the model and color member variables are private,(data hiding) meaning they can only be accessed within the Car class. The start() and stop() methods are public, allowing external code to interact with the Car object. They all are encapsulated.

```
class Car {
private:
    string model;
    string color;
public:
    void start() {
        // Code to start the car
    }
    void stop() {
        // Code to stop the car
    }
};
```

Constructors & Destructors

- **Constructors** are special member functions of a class that are automatically called when an object of that class is created. They are used to initialize the object's state. They have the same name as the class itself. They don’t have a return type.

- **Destructors** are special member functions that are automatically called when an object is destroyed. They are used to clean up resources allocated by the object

```
class MyClass {
public:
    // Constructor
    MyClass() {
        // Initialization code
    }
    // Destructor
    ~MyClass() {
        // Cleanup code
    }
};
```

Copy Constructor

- A copy constructor is a special constructor used to create a new object as a copy of an existing object. It is invoked when an object is passed by value or when an object is explicitly copied.
- A copy constructor is like a photocopy machine for objects. It creates a duplicate of an existing object, so we have two identical copies.
- Imagine you have a car, and you want another car that's exactly the same. A copy constructor helps you make that duplicate, saving you time and effort.

```
class Point {
private:
    int x;
    int y;
public:
    // Copy Constructor
    Point(const Point& p) {
        x = p.x;
        y = p.y;
    }
};
```

Default and No-Argument Constructors

- A default constructor is a constructor with no parameters. It is automatically invoked when an object is created without any arguments.
- A no-argument constructor is a constructor with parameters but with default values. It is used to provide default initialization for objects.
- Default constructors are like **pre-made templates for objects**. They allow us to create objects without specifying any initial values. It's like ordering a standard pizza without any extra toppings.
- No-argument constructors are similar**, but they let us provide default values** for object initialization. It's like ordering a pizza and specifying that you want cheese and tomato sauce by default.

```cpp
class Rectangle {
private:
    int width;
    int height;
public:
    // Default Constructor
    Rectangle() {
        width = 0;
        height = 0;
    }

    // No-Argument Constructor with Default Values
    Rectangle(int w = 0, int h = 0) {
        width = w;
        height = h;
    }
};
```

Setters & Getters

- Setters and getters are methods used to set and get the values of private member variables, respectively. They provide controlled access to the private data of a class.
- Setters and getters are like the caretakers of our data. They help us control **how others interact with our class's variables**.
- Setters let us change the values of private variables safely. It's like giving someone permission to update the details on a form.
- Getters allow us to retrieve the values of **private** variables without directly accessing them. It's like asking for information from a trusted source.
- By using **setters** and **getters**, we can maintain **control** over our data and prevent accidental changes or misuse.

```
class Rectangle {
private:
    int width;
    int height;
public:
    void setWidth(int w) {
        width = w;
    }
    void setHeight(int h) {
        height = h;
    }
    int getWidth() {
        return width;
    }
    int getHeight() {
        return height;
    }
};
```