Introduction to C++ Classes

1 What are Classes in C++?

A class in C++ is a blueprint for creating objects. It encapsulates data and functions that operate on that data. Classes are a fundamental feature of object-oriented programming (OOP).

2 Why Use Classes?

Classes help in organizing complex programs by bundling data and the operations on that data together. This promotes:

- Modularity
- Code reuse
- Encapsulation
- Abstraction

3 Data and Methods

A class contains:

- **Data members** (variables to store state)
- **Member functions** or **methods** (functions to define behavior)

4 Encapsulation

Encapsulation means restricting direct access to some of an object's components. This is done using access specifiers:

- `public`: accessible from outside the class
- `private`: accessible only from within the class
- `protected`: accessible in derived classes

5   Constructors

Constructors are special member functions called automatically when an object is created. They initialize the object's data members.

6   Getters and Setters

Getters retrieve the value of private data members. Setters allow controlled modification of private data members.

7   Example Code

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

class Car {
private:
    string brand;
    int year;

public:
    // Constructor
    Car(string b, int y) {
        brand = b;
        year = y;
    }

    // Getter for brand
    string getBrand() {
        return brand;
    }

    // Setter for brand
    void setBrand(string b) {
        brand = b;
    }

    // Getter for year
    int getYear() {
        return year;
    }

    // Setter for year
    void setYear(int y) {
        year = y;
    }

    // Method to display car info
    void displayInfo() {
        cout << "Brand: " << brand << ", Year: " << year << endl;
    }
};
```

```cpp
int main() {
    Car myCar("Toyota", 2020);
    myCar.displayInfo();

    myCar.setBrand("Honda");
    myCar.setYear(2022);
    myCar.displayInfo();

    return 0;
}
```

8   Operator Overloading

Operator overloading allows you to redefine how operators work with user-defined types (e.g., classes). This enables natural syntax when working with custom objects like fractions, complex numbers, vectors, etc.

```cpp
#include <iostream>

class Fraction {
  int numerator;
  int denominator;

public:
  Fraction(int num = 0, int den = 1) : numerator(num),
  denominator(den) {}

  // Overload +
  Fraction operator+(const Fraction& other) const {
    int num = numerator * other.denominator + other.numerator *
  denominator;
    int den = denominator * other.denominator;
    return Fraction(num, den);
  }

  // Overload -
  Fraction operator-(const Fraction& other) const {
    int num = numerator * other.denominator - other.numerator *
  denominator;
    int den = denominator * other.denominator;
    return Fraction(num, den);
  }

  void display() const {
    std::cout << numerator << "/" << denominator << "\n";
  }
};

// Demo
int main() {
  Fraction f1(1, 3), f2(1, 6);
  Fraction sum = f1 + f2;
  Fraction diff = f1 - f2;

  std::cout << "Sum: ";
```

```cpp
    sum.display();
    std::cout << "Difference: ";
    diff.display();

    return 0;
}
```

9   Conclusion

Classes are essential for building scalable and maintainable C++ applications.
They provide structure and abstraction through data hiding and encapsulation.