Object-Oriented Programming (OOP) in C++

# 1. Introduction to OOP

Object-Oriented Programming (OOP) is a programming paradigm based on the concept of 'objects', which can contain data and code to manipulate that data. It provides a clear modular structure for programs and allows objects to be modeled based on real-world entities. C++ is a powerful object-oriented language that offers features such as classes, inheritance, and polymorphism.

# 2. Key Concepts of OOP

## 2.1 Classes and Objects

A class is a blueprint for creating objects. It defines a datatype by bundling data and methods that work on the data into one single unit. An object is an instance of a class.

Example:

class Animal {  
public:  
 string name;  
 int age;  
   
 void setValues(string nm, int ag) {  
 name = nm;  
 age = ag;  
 }  
   
 void display() {  
 cout << "Name: " << name << ", Age: " << age << endl;  
 }  
};  
  
int main() {  
 Animal dog;  
 dog.setValues("Buddy", 5);  
 dog.display();  
 return 0;  
}

## 2.2 Encapsulation

Encapsulation is the mechanism of bundling data and the methods that operate on that data within a single unit, or class. It restricts direct access to some of an object's components, which can prevent the accidental modification of data.

Example:

class EncapsulatedAnimal {  
private:  
 string name;  
 int age;  
   
public:  
 void setValues(string nm, int ag) {  
 name = nm;  
 age = ag;  
 }  
   
 void display() {  
 cout << "Name: " << name << ", Age: " << age << endl;  
 }  
};  
  
int main() {  
 EncapsulatedAnimal cat;  
 cat.setValues("Whiskers", 3);  
 cat.display();  
 return 0;  
}

## 2.3 Inheritance

Inheritance is a feature that represents the 'is-a' relationship between different classes. It allows a class to inherit attributes and methods from another class.

Example:

class Animal {  
public:  
 string name;  
 void setName(string nm) {  
 name = nm;  
 }  
 void display() {  
 cout << "Name: " << name << endl;  
 }  
};  
  
class Dog : public Animal {  
public:  
 void bark() {  
 cout << name << " says Woof!" << endl;  
 }  
};  
  
int main() {  
 Dog dog;  
 dog.setName("Buddy");  
 dog.display();  
 dog.bark();  
 return 0;  
}

## 2.4 Polymorphism

Polymorphism allows methods to do different things based on the object it is acting upon. It is often expressed as 'one interface, many implementations'.

Example:

class Animal {  
public:  
 virtual void makeSound() {  
 cout << "Some generic animal sound" << endl;  
 }  
};  
  
class Dog : public Animal {  
public:  
 void makeSound() override {  
 cout << "Woof!" << endl;  
 }  
};  
  
class Cat : public Animal {  
public:  
 void makeSound() override {  
 cout << "Meow!" << endl;  
 }  
};  
  
int main() {  
 Animal\* animal;  
 Dog dog;  
 Cat cat;  
   
 animal = &dog;  
 animal->makeSound();  
   
 animal = &cat;  
 animal->makeSound();  
   
 return 0;  
}

## 2.5 Abstraction

Abstraction is the concept of hiding the complex implementation details and showing only the essential features of an object. It helps in reducing programming complexity and effort.

Example:

class AbstractAnimal {  
public:  
 virtual void sound() = 0; // Pure virtual function  
};  
  
class Dog : public AbstractAnimal {  
public:  
 void sound() override {  
 cout << "Woof!" << endl;  
 }  
};  
  
int main() {  
 Dog dog;  
 dog.sound();  
 return 0;  
}