Lập trình hướng đối tượng (OOP)

Khái niệm về OOP

Lập trình hướng đối tượng là phương pháp lập trình **lấy đối tượng làm nền tảng để xây dựng giải thuật và chương trình.**

Một số khái niệm cơ bản

Lớp (Class): là tập hợp các đối tượng có điểm tương đồng. Đối tượng là 1 thể hiện của lớp.

Đối tượng (Object): là một thực thể có trạng thái (hay thuộc tính) và hành vi (hay phương thức)

Thuộc tính (Attributes) là những thông tin, đặc điểm của đối tượng

Phương thức (Methods) là những thao tác, hành động mà đối tượng có thể thực hiện

Phạm vi truy xuất: là **khả năng truy xuất thành phần** của một lớp nào đó từ bên ngoài phạm vi lớp đó, giúp đảm bảo tính đóng gói và che giấu thông tin của đối tượng. Đối với C++, có 3 từ khóa xác định phạm vi truy xuất.

- **private:** chỉ có thể được truy cập **bên trong phạm vi lớp**
- **protected:** chỉ có thể được truy **cập bên trong phạm vi lớp** và **các lớp con kế thừa nó**
- **public:** có thể được truy cập **từ bất kì hàm nào**, dù ở trong hay ngoài lớp

Phương thức thiết lập (Constructor): khởi tạo thể hiện của lớp (khởi tạo các giá trị thành phần của đối tượng).

Phương thức hủy bỏ (Destructor): thực hiện việc dọn dẹp cần thiết trước khi một đối tượng bị hủy.

Phương thức truy vấn (Query): sử dụng tiền tố "get" hoặc "is" (đối với truy vấn điều kiện), không thay đổi trạng thái của đối tượng.

Phương thức cập nhật (Update): sử dụng tiền tố "set", thay đổi trạng thái của đối tượng.

Hàm bạn (từ khóa **friend):** là hàm không thuộc lớp (tức là không có toán tử phạm vi :: trước tên hàm) nhưng có quyền truy cập các thành viên private hoặc protected của **lớp xem nó như bạn**.

Lớp bạn: là lớp có thể truy cập các thành phần private và protected **của lớp xem nó như bạn**. Nếu lớp B là lớp bạn của lớp A, ta **cần khai báo bên trong lớp A** rằng B là bạn của nó, bằng cách thêm từ khóa **friend** trước lớp B. Mối quan hệ friend là một chiều, không có tính đối xứng (Tức là B là bạn của A, nhưng A không là bạn của B nếu B không khai báo điều đó), không có tính bắc cầu.

Sự nhập nhằng: .xảy ra khi thực hiện chuyển kiểu bằng Constructor và chuyển kiểu bằng toán tử chuyển kiểu, khiến cho trình biên dịch không xác định nên chuyển kiểu bằng cách nào, dẫn đến việc mất đi cơ chế chuyển kiểu tự động. Cách xử lý duy nhất là thực hiện chuyển kiểu tường minh (mất đi sự tiện lợi của cơ chế chuyển kiểu tự động).

Trong kế thừa, khi khởi tạo thể hiện của 1 lớp con thì thứ tự xử lý sẽ là: Gọi Constructor lớp cha → Gọi Constructor lớp con → khởi tạo thể hiện của lớp con. Đối với Destructor: Gọi Destructor lớp con → Gọi Destructor lớp cha → hủy lớp con.

Sự mơ hồ (thừa kế xung đột): xảy ra khi sử dụng phương thức **mang tên giống nhau** của 1 lớp con thừa kế từ **nhiều lớp cha khác nhau** (khi sử dụng đa kế thừa hoặc kế thừa phân cấp hoặc cả hai). Để giải quyết tình trạng này mà không cần đặt lại tên cho các phương thức, ta cần phân biệt rõ ràng chúng bằng **toán tử phạm vi (::)**

Các tính chất quan trọng

Trừu tượng (Abstraction): Trong việc thiết kế các đối tượng, **cần khái quát hóa cách nhìn** và rút ra **những đặc trưng chung cần quan tâm.** (bỏ qua các chi tiết không cần thiết)

Tính chất này thường xác định trong các lớp giao diện (interface) hay các lớp cơ sở trừu tượng.

Đóng gói (Encapsulation): Các dữ liệu hoặc phương thức liên quan với nhau được **đóng gói thành các lớp** để tiện cho việc quản lý và sử dụng, ngoài ra còn để **che giấu thông tin** và chi tiết cài đặt nội bộ không thể nhìn thấy.

Tính chất này đảm bảo sự toàn vẹn của đối tượng.

Thừa kế (Inheritance): Một đối tượng (con) có tất cả trạng thái và hành vi của đối tượng khác (cha).

Bao gồm kế thừa đơn và đa kế thừa. Ngoài ra còn có kế thừa đa cấp, kế thừa phân cấp và kế thừa ảo

Cho phép cài đặt nhiều quan hệ giữa các đối tượng. Lớp con có thể trở thành lớp cha cho các lớp con khác.

Có 3 loại thừa kế: private, protected và public (loại thừa kế cần được khai báo).

public: public của cha → public ở con; protected của cha → protected ở con.

protected: public và protected của cha → protected ở con

private: public và protected của cha → private ở con

Việc truy cập các thành viên được kế thừa của lớp dẫn xuất (lớp con) phụ thuộc vào **phạm vi truy xuất** của các thành viên đó, đồng thời cũng phụ thuộc **loại thừa kế** của lớp con

Nếu mọi Constructor của lớp cha đều cần cung cấp tham số thì lớp con bắt buộc phải có Constructor để cung cấp các tham số đó.

Lớp con có thể viết lại (override) các phương thức đã định nghĩa của lớp cha.

Lớp con không cần và không được thực hiện các thao tác dọn dẹp cho các thành phần thuộc lớp cha.

Con trỏ trỏ đến đối tượng thuộc lớp cha thì có thể trỏ đến các đối tượng thuộc lớp con, nhưng điều ngược lại không đúng (sử dụng ép kiểu có thể thực hiện điều ngược lại, nhưng thao tác này có thể nguy hiểm)

Đa hình (Polymorphism): Là cơ chế cho phép một thao tác hay thuộc tính có thể được định nghĩa tại nhiều lớp và có thể có nhiều cách cài đặt khác nhau ở mỗi lớp.

Tính đa hình có 2 cách thể hiện:

- Qua con trỏ (pointer) và hàm ảo (virtual funtion)

- Qua nạp chồng hàm (funtion overloading) và nạp chồng toán tử (operator overloading)

Phương thức ảo (từ khóa **virtual**) giải quyết vấn đề **con trỏ** thuộc lớp cha trỏ đến lớp con và mong muốn dùng con trỏ đó truy xuất hàm thành phần định nghĩa tại lớp con. Một phương thức ảo có thể là friend trong một lớp khác, nhưng các hàm friend của lớp không thể là phương thức ảo. Chỉ có Destructor mới có thể được đánh dấu là hàm ảo (trở thành hàm hủy ảo), còn Constructor thì không.

Nạp chồng hàm: cho phép khai báo và sử dụng cùng 1 tên gọi cho các hàm có cùng mục đích nhưng **khác nhau về kiểu dữ liệu** và **tham số đầu vào**.

Nạp chồng toán tử (từ khóa **operator**): định nghĩa toán tử có sẵn phục vụ cho dữ liệu riêng do bạn tạo ra. Có một số hạn chế như không thể tạo toán tử mới, không thay đổi thứ tự ưu tiên, không tạo cú pháp mới và không định nghĩa lại toán tử có sẵn. Một số toán tử cần ràng buộc ý nghĩa (tức phải định nghĩa **hàm thành phần** với **số tham số ít hơn số ngôi là 1**), cũng như tôn trọng ý nghĩa gốc của nó.