Kế thừa và đa hình trong C#

1.Kế thừa (Interface)

Kế thừa (inheritance) là một khái niệm trong lập trình hướng đối tượng, cho phép một lớp (class) có thể kế thừa các thuộc tính và phương thức từ một lớp khác. Điều này giúp tái sử dụng mã và xây dựng các cấu trúc phân cấp lớp một cách hiệu quả. Lớp kế thừa được gọi là lớp dẫn xuất (derived class), và lớp được kế thừa là lớp cơ sở (base class).

Cách thức kế thừa giúp tái sử dụng mã nguồn và giảm thiểu trùng lặp:

Tái sử dụng các phương thức và thuộc tính chung:

Khi một lớp dẫn xuất kế thừa từ lớp cơ sở, nó có thể sử dụng lại tất cả các phương thức và thuộc tính mà lớp cơ sở đã định nghĩa mà không cần phải viết lại những đoạn mã tương tự. Điều này giúp tránh lặp lại mã (code duplication) trong nhiều lớp khác nhau.

Giảm sự trùng lặp trong cấu trúc mã:

Nếu không có kế thừa, các lớp với các chức năng tương tự sẽ phải lặp lại các phương thức và thuộc tính giống nhau. Khi có kế thừa, các thuộc tính chung chỉ cần định nghĩa một lần trong lớp cơ sở, và các lớp dẫn xuất có thể tự động sử dụng chúng

Dễ dàng bảo trì và mở rộng:

Kế thừa cho phép dễ dàng cập nhật mã nguồn, bởi vì nếu có thay đổi trong phương thức hoặc thuộc tính chung (có trong lớp cơ sở), chỉ cần sửa đổi một nơi duy nhất (trong lớp cơ sở), và các lớp dẫn xuất tự động kế thừa sự thay đổi đó.

Khi thêm một lớp mới kế thừa từ lớp cơ sở, lớp mới có thể tận dụng lại các tính năng có sẵn mà không cần tái tạo từ đầu.

Tính mở rộng linh hoạt:

Lớp dẫn xuất không chỉ sử dụng lại mã của lớp cơ sở mà còn có thể ghi đè (override) hoặc mở rộng các phương thức, giúp tạo ra các hành vi cụ thể hơn mà không làm thay đổi lớp cơ sở.

Lợi ích của việc kế thừa:

Tiết kiệm thời gian và công sức: Lập trình viên không cần viết lại mã đã có, giúp tập trung vào các tính năng cụ thể hơn.

Dễ bảo trì: Thay đổi ở một nơi (lớp cơ sở) sẽ ảnh hưởng đến tất cả các lớp dẫn xuất, giúp mã dễ quản lý và giảm lỗi.

Tăng tính mở rộng: Các lớp có thể dễ dàng mở rộng mà không làm thay đổi mã cốt lõi.

2.Tính đa hình

Tính đa hình (Polymorphism) trong C# là một trong bốn đặc tính quan trọng của lập trình hướng đối tượng (OOP), cùng với tính đóng gói (encapsulation), tính kế thừa (inheritance), và tính trừu tượng (abstraction). Tính đa hình cho phép một đối tượng có thể được sử dụng như nhiều dạng khác nhau

Có hai loại đa hình trong C#:

1. Đa hình tại thời gian biên dịch (Compile-time polymorphism) hay còn gọi là đa hình tĩnh: Thường được thực hiện thông qua nạp chồng phương thức (method overloading) và nạp chồng toán tử (operator overloading).
2. Đa hình tại thời gian chạy (Runtime polymorphism) hay còn gọi là đa hình động: Thực hiện thông qua ghi đè phương thức (method overriding). Nó sử dụng tính kế thừa và các lớp dẫn xuất để cung cấp các hành vi khác nhau cho các phương thức.

Phân biệt giữa Đa hình tĩnh (Compile-time Polymorphism) và Đa hình động (Run-time Polymorphism)

1. Đa hình tĩnh (Compile-time Polymorphism):

* Định nghĩa: Đa hình tĩnh xảy ra khi quyết định phương thức nào sẽ được gọi được thực hiện trong quá trình biên dịch. Điều này thường được thực hiện thông qua nạp chồng phương thức (Method Overloading) hoặc nạp chồng toán tử (Operator Overloading).
* Cơ chế: Việc lựa chọn phương thức hoặc toán tử phụ thuộc vào số lượng, kiểu dữ liệu của tham số đầu vào.
* Ưu điểm: Chạy nhanh hơn vì tất cả các quyết định được thực hiện trong quá trình biên dịch.
* Ví dụ: Nạp chồng phương thức với các tham số khác nhau.
* Từ khóa: Không cần từ khóa đặc biệt, chỉ cần các phương thức có cùng tên nhưng khác nhau về tham số.

2. Đa hình động (Run-time Polymorphism):

* Định nghĩa: Đa hình động xảy ra khi quyết định phương thức nào sẽ được gọi được thực hiện trong quá trình chạy. Điều này được thực hiện thông qua ghi đè phương thức (Method Overriding).
* Cơ chế: Một lớp con có thể ghi đè một phương thức của lớp cha bằng cách sử dụng từ khóa override. Lúc chạy, đối tượng cụ thể sẽ quyết định phương thức nào được gọi.
* Ưu điểm: Tính linh hoạt cao, dễ mở rộng.
* Ví dụ: Ghi đè phương thức từ lớp cơ sở trong các lớp con.
* Từ khóa: Sử dụng từ khóa virtual trong phương thức lớp cơ sở và override trong lớp con

3.So sánh và ứng dụng

1. Kế thừa (Inheritance):

* Định nghĩa: Kế thừa là khả năng một lớp con có thể mở rộng hoặc tái sử dụng thuộc tính và phương thức của lớp cha. Lớp con kế thừa từ lớp cha bằng từ khóa :, và có thể sử dụng lại mã nguồn của lớp cha, đồng thời bổ sung các đặc tính riêng.
* Đặc điểm:
  + Cho phép sử dụng lại mã nguồn.
  + Quan hệ "IS-A" giữa lớp cha và lớp con. Ví dụ: Một Oto là một PhuongTien.
  + Lớp con có thể kế thừa thuộc tính và phương thức của lớp cha, đồng thời thêm các thuộc tính hoặc phương thức mới.
  + Từ khóa: class và :.
* Ứng dụng thực tế:
  + Giúp quản lý và tổ chức mã nguồn theo cách rõ ràng hơn.
  + Giảm thiểu việc trùng lặp mã bằng cách tái sử dụng các phương thức và thuộc tính đã được định nghĩa trong lớp cha.

2. Đa hình (Polymorphism):

* Định nghĩa: Đa hình là khả năng của một đối tượng có thể thay đổi hình thức khi sử dụng. Đa hình cho phép một phương thức có thể có nhiều cách triển khai khác nhau, dựa trên ngữ cảnh hoặc loại đối tượng cụ thể.
* Đặc điểm:
  + Đa hình giúp thay đổi cách một phương thức hoạt động dựa trên lớp dẫn xuất.
  + Cho phép sử dụng một giao diện chung cho nhiều đối tượng khác nhau.
  + Có thể là đa hình tĩnh (nạp chồng phương thức) hoặc đa hình động (ghi đè phương thức).
  + Từ khóa: virtual, override, new, abstract.
* Ứng dụng thực tế:
  + Tăng tính linh hoạt và dễ mở rộng của mã nguồn.
  + Cải thiện khả năng quản lý các đối tượng phức tạp có cùng kiểu cha nhưng có hành vi khác nhau.

Ứng dụng thực tế của Kế thừa và Đa hình trong phát triển phần mềm:

Kế thừa:

1. Tổ chức phân cấp lớp: Trong một hệ thống quản lý, có thể tạo ra một cấu trúc phân cấp lớp như NhanVien, NhanVienToanThoiGian, NhanVienBanThoiGian,... Các lớp con có thể kế thừa các thuộc tính và phương thức của lớp cha NhanVien.
2. Tái sử dụng mã: Giúp tái sử dụng mã mà không cần viết lại nhiều lần. Ví dụ, lớp Xe có thể kế thừa các thuộc tính và phương thức từ lớp PhuongTien.

Đa hình:

1. Sử dụng giao diện chung: Đa hình cho phép triển khai một giao diện chung, ví dụ như trong hệ thống thanh toán, có thể có các lớp TheTinDung, ViDienTu kế thừa từ PhuongThucThanhToan, mỗi lớp sẽ có cách thanh toán riêng.
2. Xử lý khác nhau cho các đối tượng khác nhau: Các đối tượng khác nhau của cùng một kiểu có thể có hành vi khác nhau. Ví dụ: Oto và XeMay đều là PhuongTien, nhưng chúng có cách thức di chuyển khác nhau khi gọi DiChuyen().