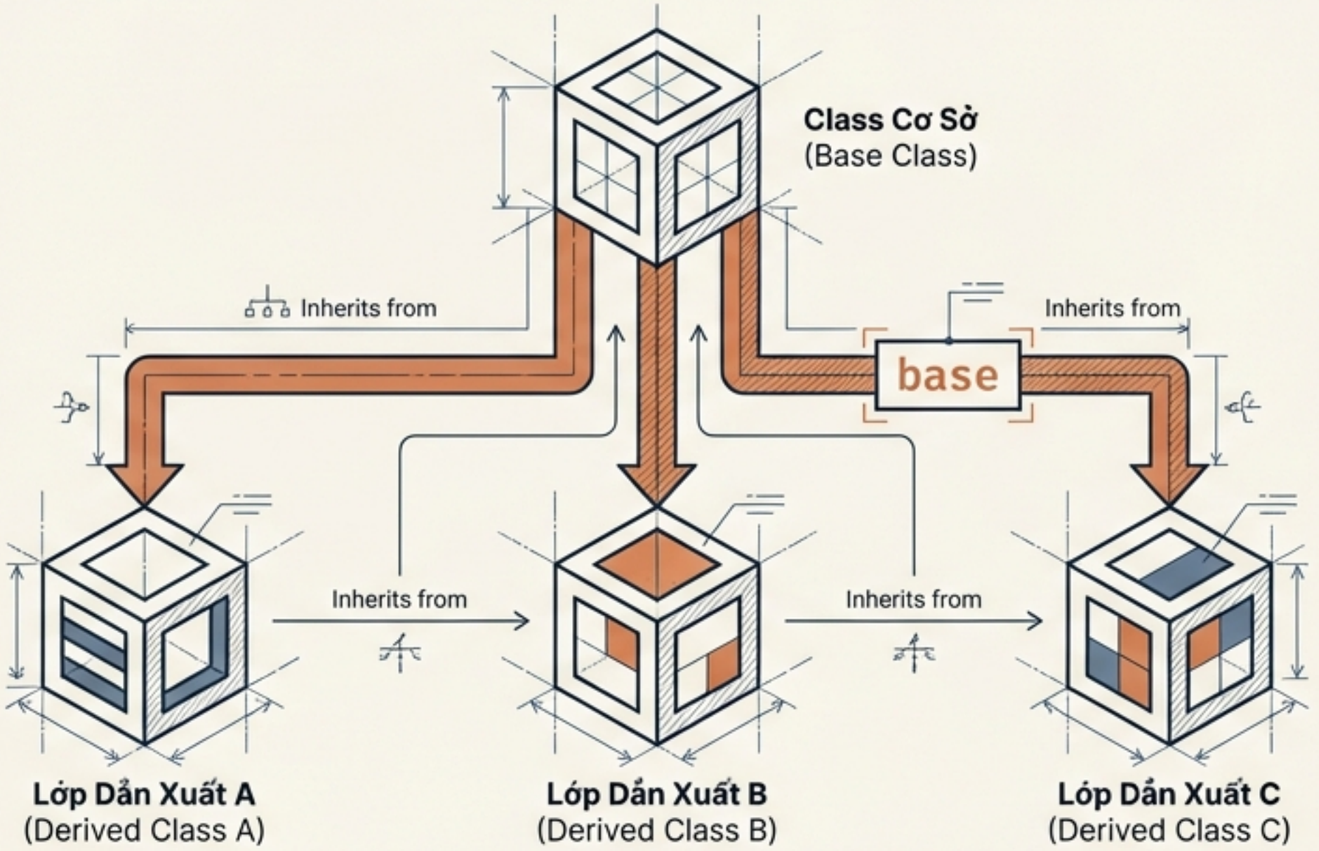
Làm chủ Kế Thừa (Inheritance) và Từ Khóa base trong C#

Giải quyết triệt để vấn đề lặp code và xây dựng kiến trúc phần mềm linh hoạt.



Lặp lại code là kẻ thù số một của bảo trì phần mềm

```
class NhanVienVanPhong {
  string Ten; int LuongCoBan;
}
```

```
class NhanVienSanXuat {
  string Ten; int LuongCoBan;
}
```

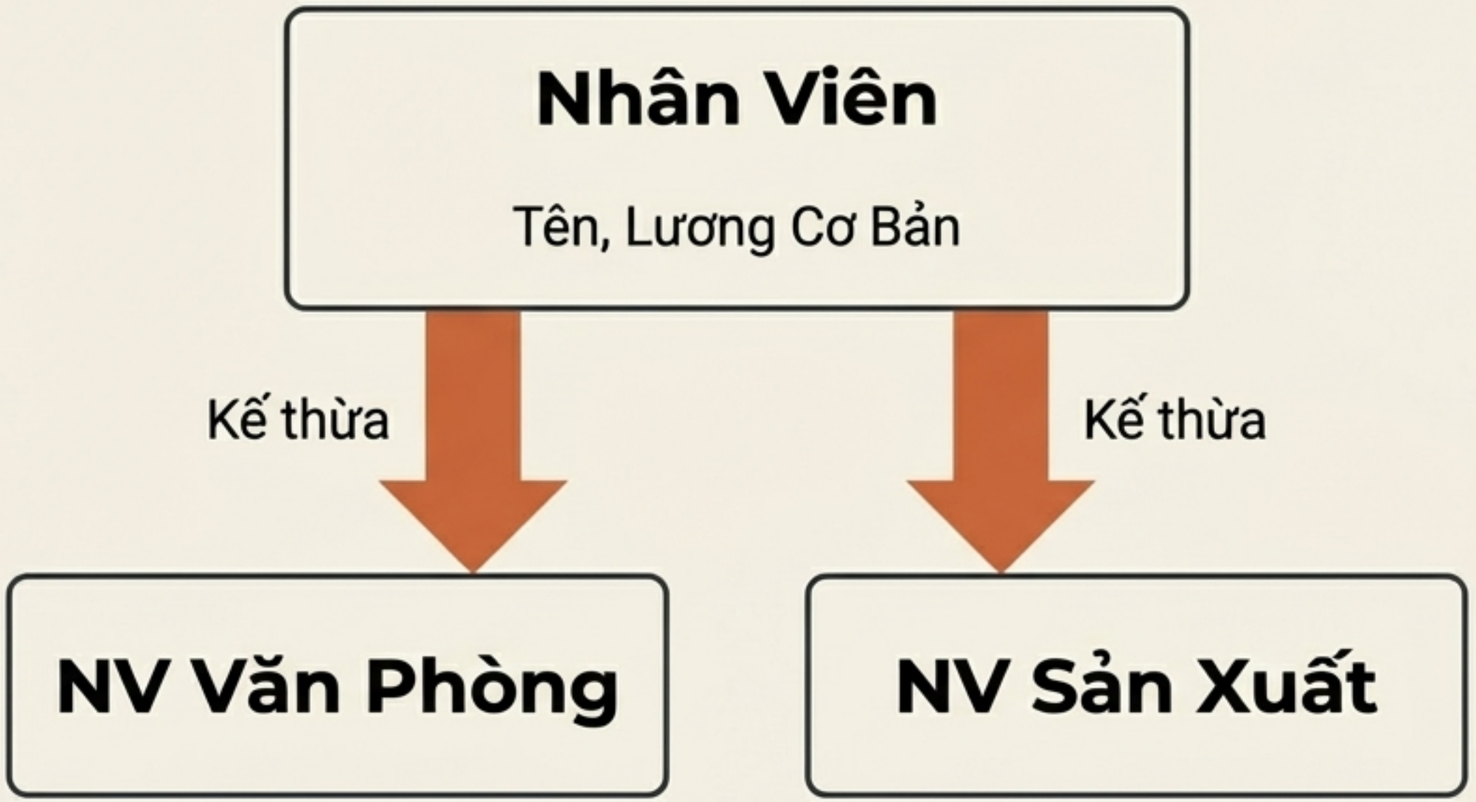
```
class NhanVienQuanLy {
  string Ten; int LuongCoBan;
}
```

Giả sử có hệ thống quản lý nhân sự gồm 3 đối tượng: Nhân viên văn phòng, Nhân viên sản xuất, Nhân viên quản lý. Tất cả đều chia sẻ các thuộc tính: Tên, Lương cơ bản, Hiển thị thông tin.

- Code lặp lại
- Khó bảo trì
- Dễ sai sót khi chỉnh sửa

Lớp con kế thừa trọn vẹn đặc tính của lớp cha

Kế thừa sinh ra để giải quyết sự lặp lại. Class con nhận lại toàn bộ thuộc tính và phương thức của class cha. Giúp mã nguồn gọn gàng, có tổ chức và dễ dàng mở rộng trong tương lai.



Thiết kế lớp cha chuẩn mực để dự phòng cho tương lai

Tư duy thiết kế cốt lõi: Những gì có khả năng thay đổi thì phải để virtual.

- **protected**: Đảm bảo tính đóng gói, nhưng vẫn cho phép class con truy cập trực tiếp.
- **virtual**: Cấp quyền cho phép class con thay đổi phương thức (ví dụ: TinhLuong()).

```
class NhanVien {
  protected string Ten;
  protected int LuongCoBan;

  public virtual void TinhLuong() {
    // Logic mặc định
  }
}
```

Trình tự khởi tạo sinh mệnh đối tượng với base()

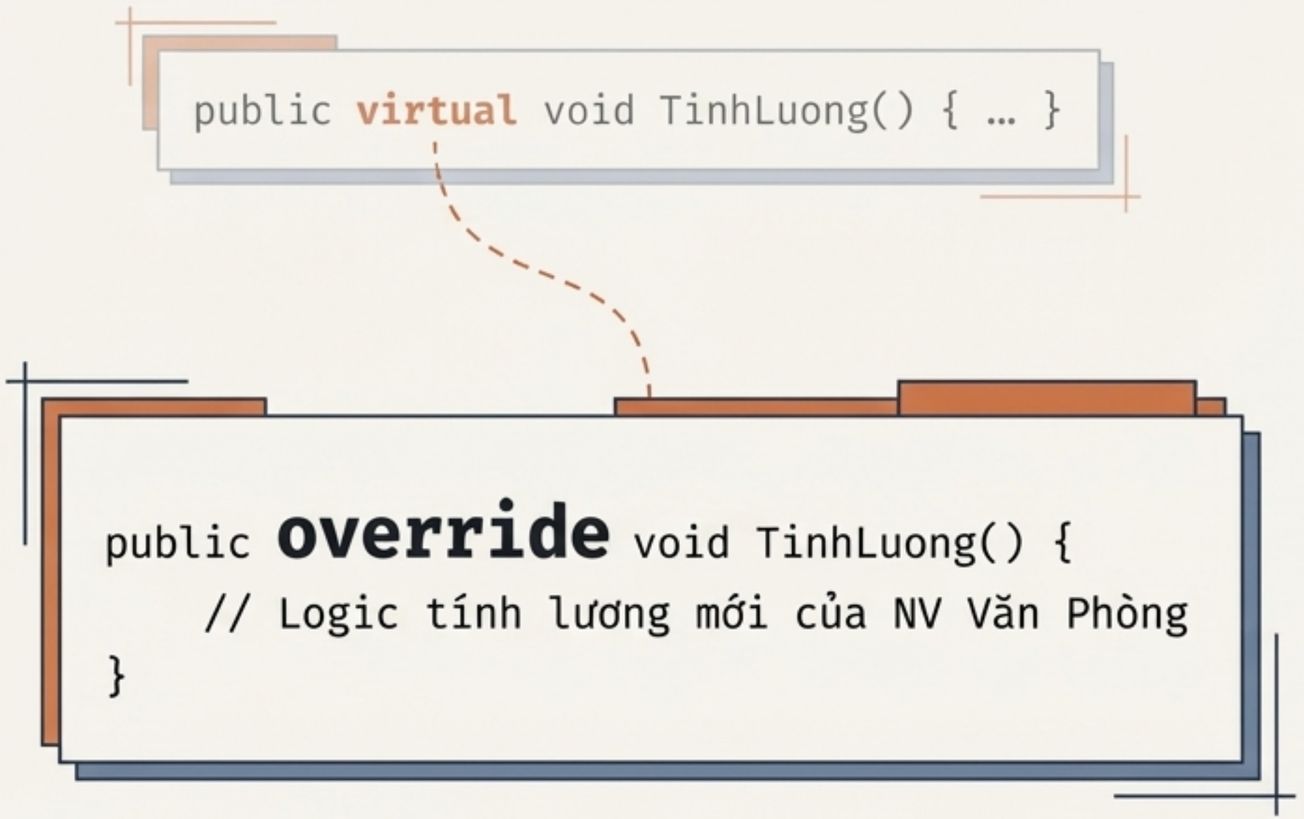
Nguyên tắc: Constructor của lớp Cha luôn phải chạy trước lớp Con.
Lỗi thường gặp: Nếu lớp cha không có constructor rỗng mà lớp con không gọi base -> Lỗi compile ngay lập tức.


```
public NhanVienVanPhong(string ten, int luongCoBan) : base(ten, luongCoBan) {
    // Khởi tạo phần mở rộng của lớp con
}
```

Đảm bảo dữ liệu cốt lõi được khởi tạo đúng cách

Tiến hóa hành vi đối tượng bằng từ khóa override

Mỗi loại nhân viên có một công thức tính lương riêng biệt.
Sử dụng **override** để ghi đè phương thức TinhLuong() đã được khai báo virtual ở lớp cha.

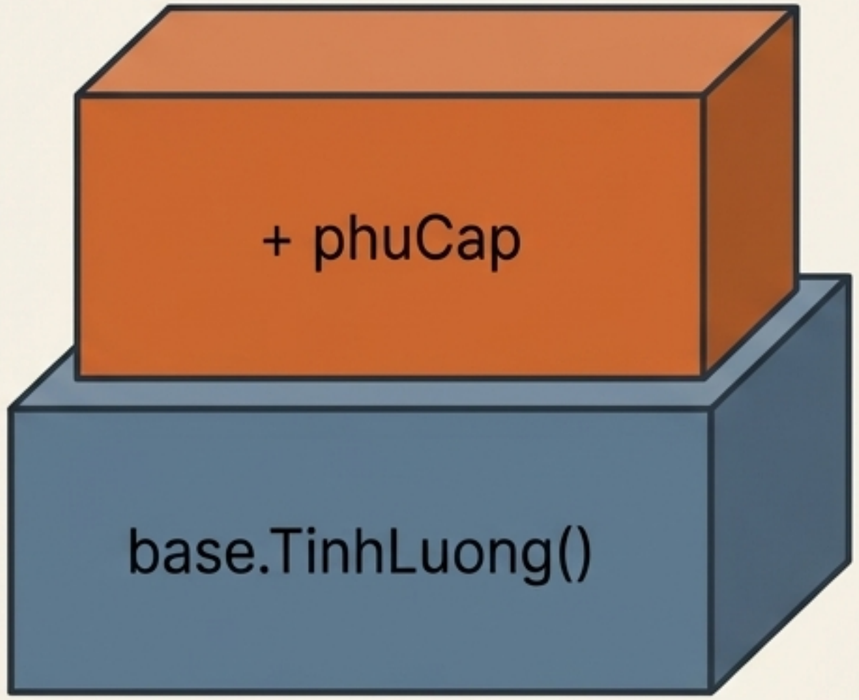


Mở rộng tính năng không có nghĩa là viết lại từ đầu

- Sử dụng base.TinhLuong() để tận dụng lại logic đã có của lớp cha.

- Không lặp lại code của cha. Nếu sau này cha thay đổi cách tính lương cơ bản, lớp con vẫn hoạt động đúng.

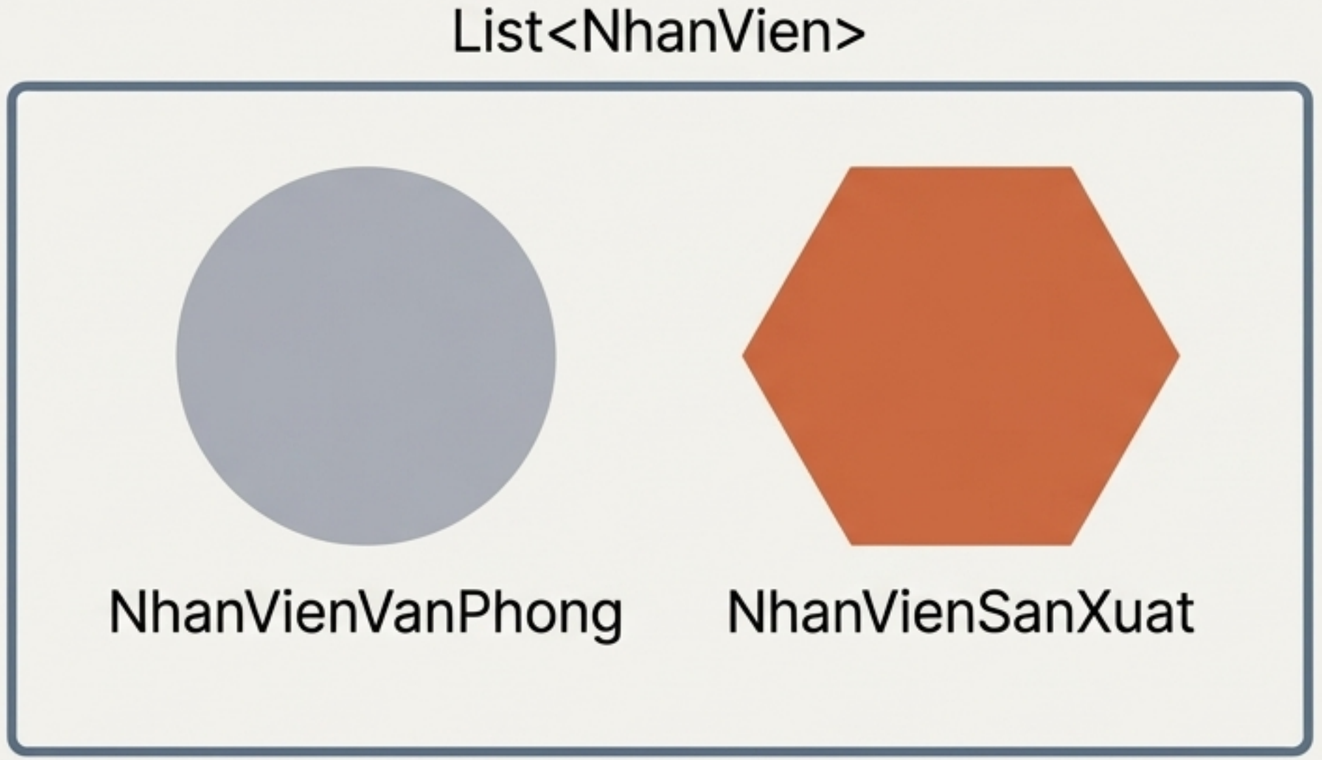
- Đây là hành động mở rộng an toàn, không phá vỡ cấu trúc hiện tại.




```
return base.TinhLuong() + phuCap;
```

Chìa khóa mở ra sức mạnh thực sự của Đa Hình

Điểm mấu chốt: Một danh sách kiểu Cha hoàn toàn có thể chứa các object kiểu Con. Đây chính là nền tảng cốt lõi của tính Đa hình (Polymorphism).



Danh sách kiểu cha, nhưng chứa object con.

Ứng dụng mô hình kế thừa vào phát triển Game

Lớp Cha: Monster (Có máu, tốc độ di chuyển cơ bản).

Lớp Con: Boss (Kế thừa từ Monster, nhưng ghi đè hành vi tấn công đặc biệt).

Khi hệ thống gọi hàm tấn công, hành vi của Boss sẽ tự động kích hoạt.
Đây là bước đệm hoàn hảo cho Đa hình runtime.

class Monster **class Boss : Monster**

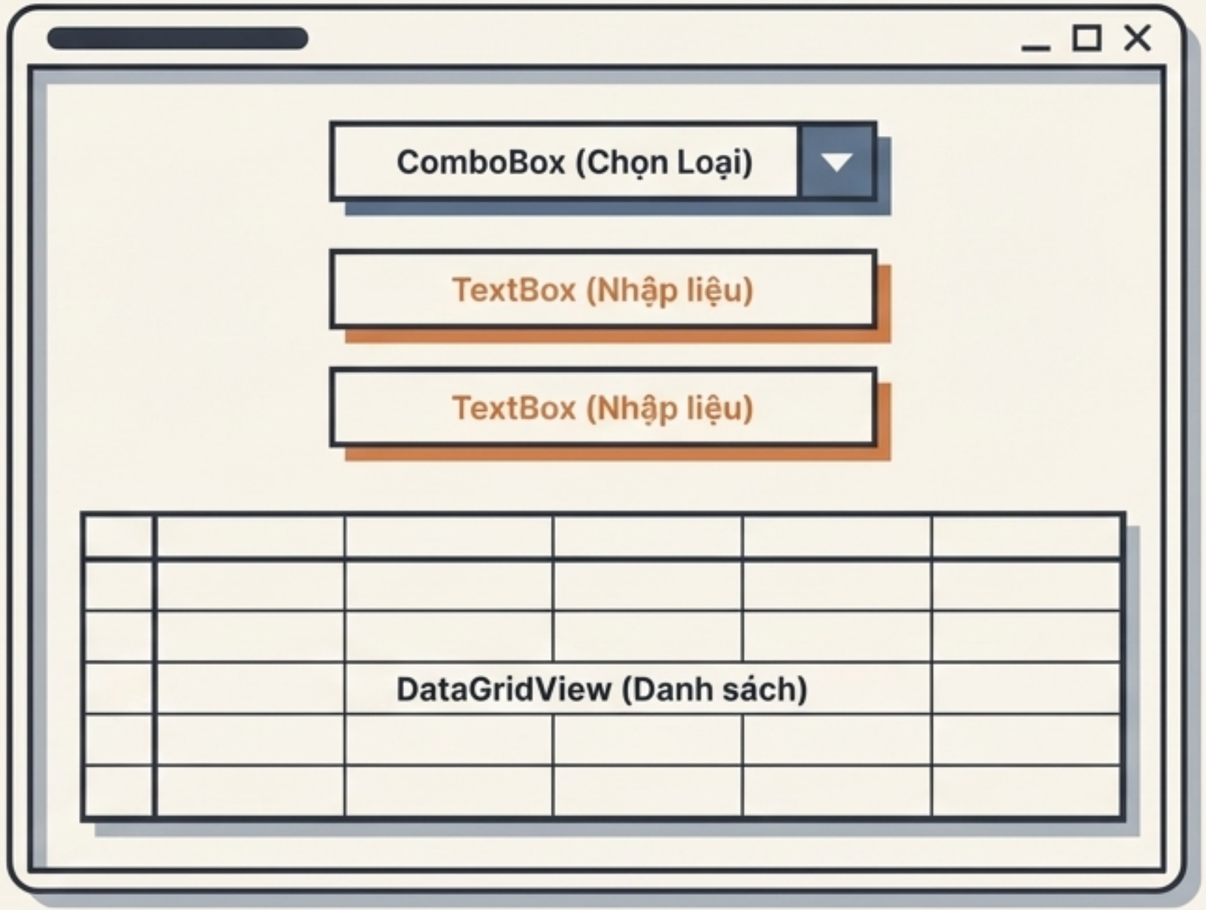
```
Boss finalBoss = new Boss();

finalBoss.Attack(); // Gọi đòn tấn công đặc biệt đã được override
```

Hiện thực hóa bằng ứng dụng WinForms chuyên nghiệp

Hệ thống thêm nhân viên đa loại với kiến trúc chuẩn mực.

Giao diện sử dụng ComboBox để chọn loại nhân viên, TextBox để nhập liệu, và DataGridView để hiển thị toàn bộ danh sách.

Logic lõi phía sau chính là mô hình kế thừa đã xây dựng, giúp việc thêm mới loại nhân viên trong tương lai cực kỳ dễ dàng.



Những trụ cột quan trọng của Kế thừa cần ghi nhớ

- `Kế thừa`: Giải pháp tuyệt đối để tái sử dụng code.
- `protected`: Giữ bí mật với bên ngoài, nhưng cởi mở với lớp con.
- `base()`: Đảm bảo gốc rễ (lớp cha) được khởi tạo đúng trình tự.
- `base.Method()`: Mở rộng hành vi mà không cần viết lại từ đầu.
- `virtual + override`: Nền tảng thiết yếu để tiến lên Đa hình (Polymorphism).

Khám phá thêm về this, static, và Phạm Vi Truy Cập tại Error404-Labs.