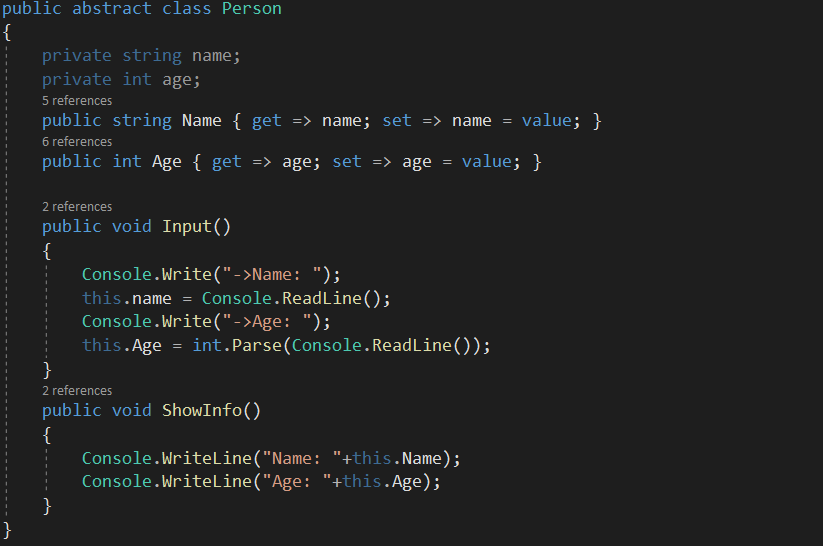
**BÁO CÁO ĐỀ TÀI**

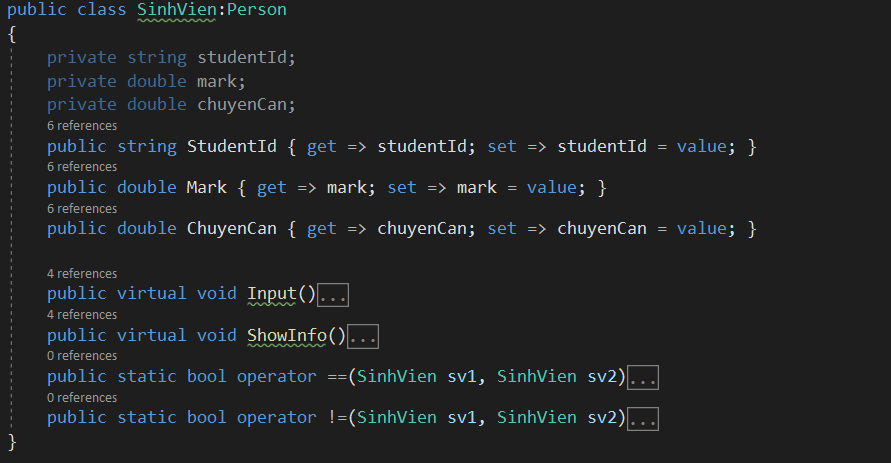
***Đề tài: Quản lí trường học***

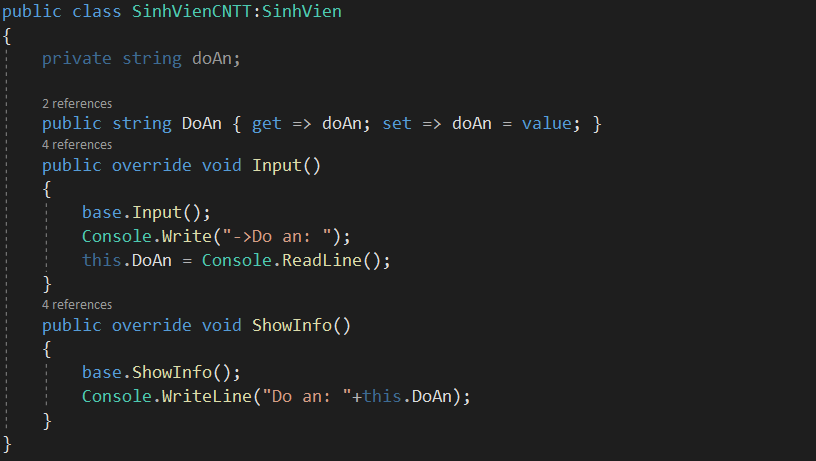
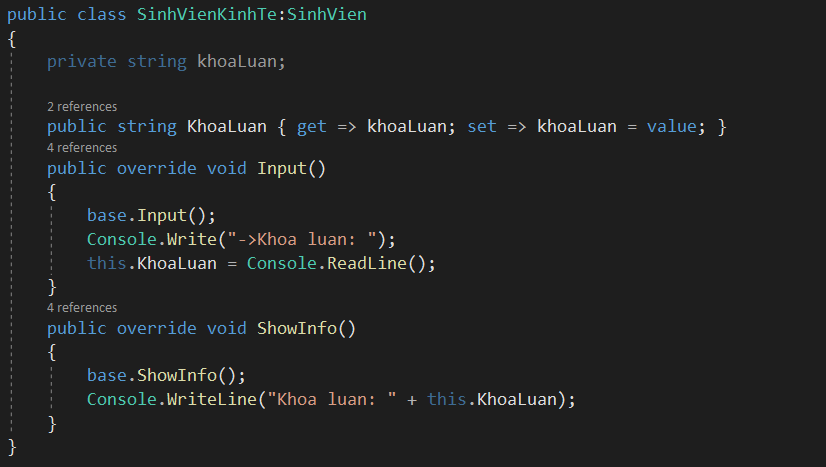
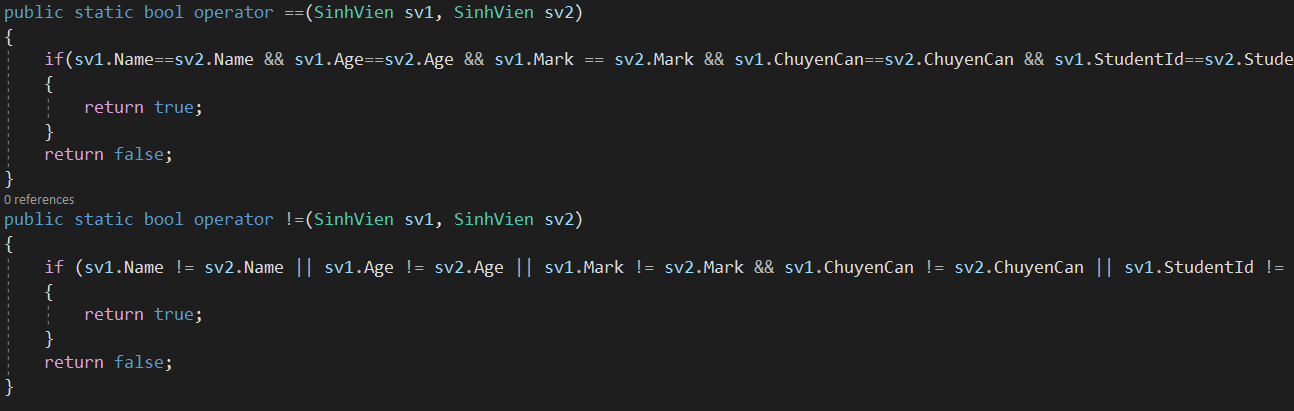
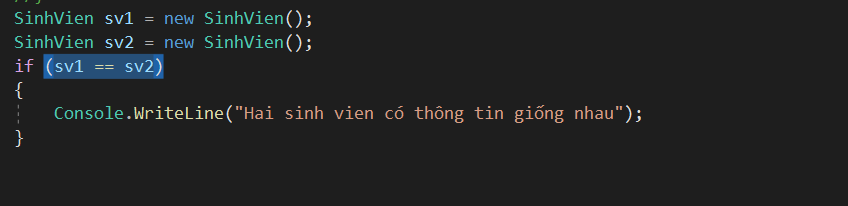
**Mô tả**: Chương trình được xây dựng để quản lí các thông tin cơ bản của cán bộ công nhân viên (Giảng viên) và Sinh viên trong đơn vị trường học.

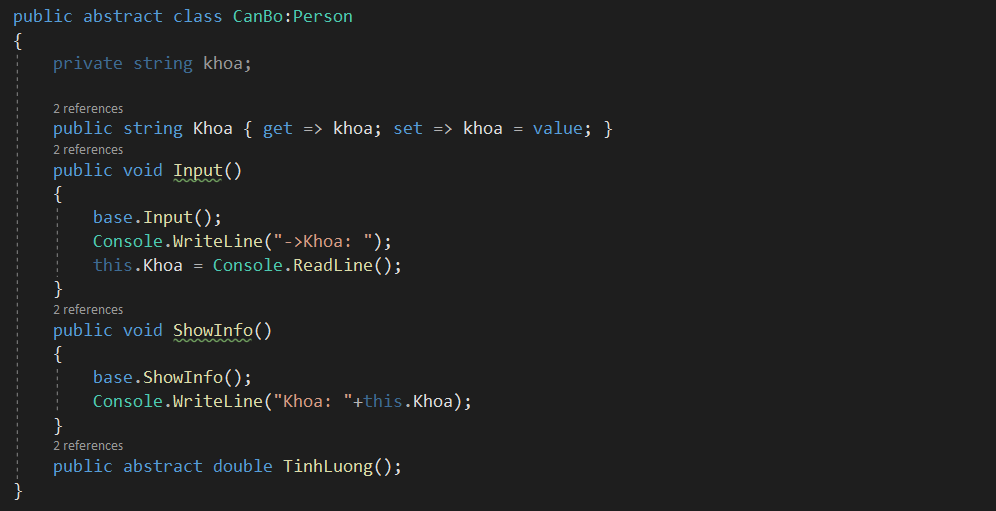
* Mỗi Cán bộ hay Sinh viên đều có những thuộc tính chung của 1 người như: tên, tuổi, giới tính, ngày sinh, quê quán….Từ đó làm nổi bật lên tính “**đặc biệt hóa-tổng quát hóa**” của các lớp(class) trong chương trình. Dựa trên điều này và tính chất trừu tượng trong Lập trình hướng đối tượng, ta xây dựng lớp **Person** chứa những thuộc tính(properties) và các phương thức(method) chung nhất.



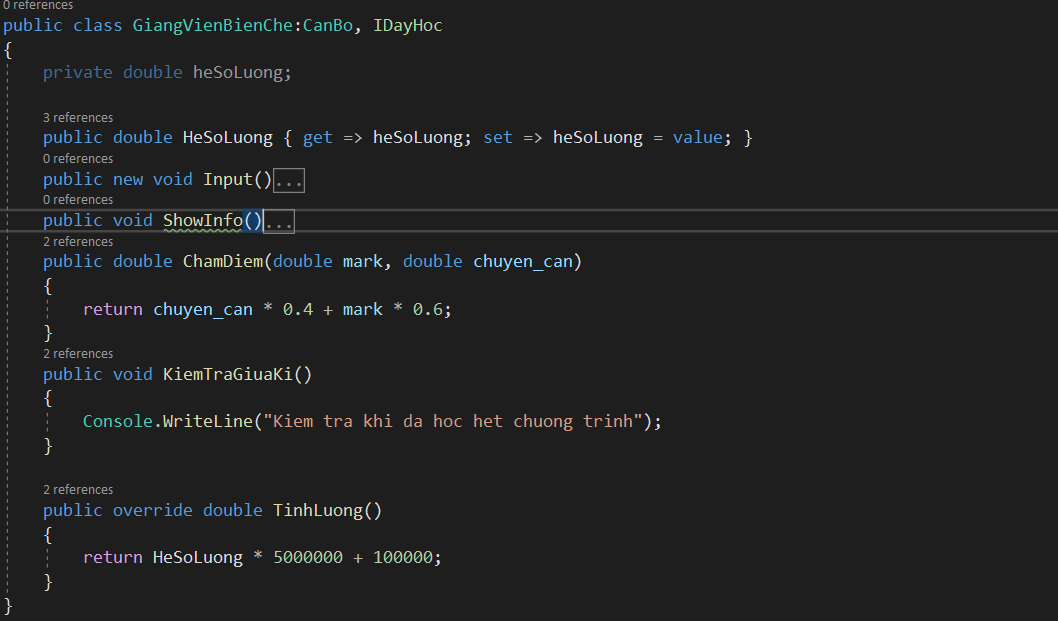
* Ở lớp **Person**, chúng ta có thể thấy các thuộc tính **name** và **age** có phạm vi truy cập(access modifier) là private. Điều này thể hiện tính đóng gói – một trong 4 tính chất của lập trình hướng đối tượng. Có thể gói dữ liệu(datas, diễn biến, trạng thái) và mã chương trình(code, phương thức) vào trong 1 class để dễ dàng quản lí. Trong class, có các thành phần không tiện cho người bên ngoài truy cập nên ta sẽ che giấu thông tin đi mà chỉ để lộ các phương thức.



* Lớp **SinhVien** mang những đặc tính chung đã được khai báo trong lớp **Person**, đồng thời lớp này còn có thêm các thuộc tính riêng mà chỉ lớp này mới có(StudentID, mark, chuyenCan). Mặc dù không khai báo những thuộc tính như name và age, nhưng lớp **SinhVien** vẫn mang các thuộc tính này do được kế thừa từ lớp **Person**. Tuy nhiên, lớp **SinhVien** không thể truy cập trực tiếp các thuộc tính của lớp cha(**Person**) do tính đóng gói đã được đề cập ở trên. Để có thể truy cập được các thuộc tính của lớp cha, thì ở lớp **Person** cần định nghĩa thêm phương thức **accessor(get/set)**. Phương thức này là public, lớp con có thể dùng các thuộc tính được kế thừa từ lớp cha thông qua phương thức này. Còn với các phương thức public của lớp cha, lớp con hoàn toàn kế thừa và dùng được thông qua từ khóa **base**.
* Phương thức **Input()** và **ShowInfo()** của lớp này được khai báo là phương thức ảo, đánh dấu đây là phương thức có thể ghi đè. Đây chính là tính đa hình trong OOP. Tính đa hình được thể hiện hoàn thiện khi có 1 lớp dẫn xuất từ lớp **SinhVien** ghi đè hai phương thức này.
* Lớp **SinhVienCNTT** và **SinhVienKinhTe** là hai lớp dẫn xuất từ lớp **SinhVien**, nó cũng mang các thuộc tính của lớp **SinhVien** và kế thừa các phương thức của lớp cha **SinhVien**. Đặc biệt, 2 phương thức ở lớp dẫn xuất **SinhVienCNTT** đã được ghi đè thông qua từ khóa override. 
* Tính đa hình còn được thể hiện rõ ràng và đầy đủ hơn trong chương trình bởi Nạp chồng toán tử so sánh bằng(**==**) và không bằng(**!=**). Nạp chồng toán tử giúp ta có thể so sánh trực tiếp hai sinh viên như so sánh 2 số nguyên.
* Lớp **CanBo** được xây dựng như là một lớp khuôn mẫu. Lớp này được kế thừa từ lớp **Person** như các lớp khác. Lớp này còn có thêm 1 phương thức trừu tượng (abstract method) là **TinhLuong()** không được khai báo thân hàm và buộc phải ghi đè khi có 1 lớp dẫn xuất kế thừa nó.



* Lớp **GiangVienBienChe** và **GiangVienHopDong** cùng kế thừa lớp **CanBo**. Mỗi lớp lại có những thuộc tính riêng. Đặc biệt, phương thức trừu tượng **TinhLuong()** được ghi đè tại mỗi lớp. Khi phương thức này được gọi, tùy theo kiểu của đối tượng thì sẽ gọi đến phương thức **TinhLuong()** riêng của đối tượng đó.



* Đối với ngôn ngữ C#, chúng ta không thể đa kế thừa(kế thừa từ nhiều lớp cùng 1 lúc). Cả lớp **GiangVienHopDong** và **GiangVienBienChe** đều có những trạng thái như dạy học, chấm điểm, kiểm tra giữa kì….trong khi lớp **CanBo** thì lại không có những trạng thái như vậy. Do đó, interface là một giải pháp thay thế cho đa kế thừa trong chương trình.

