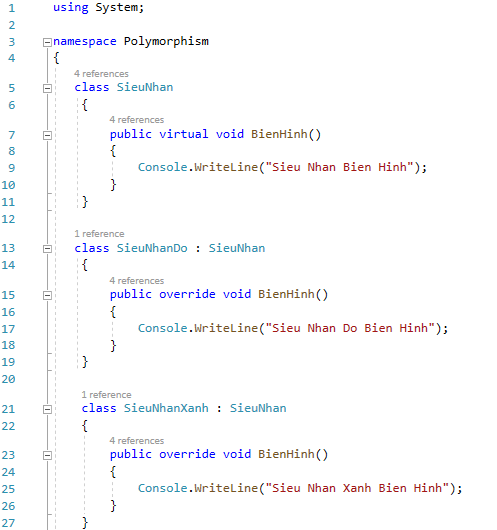
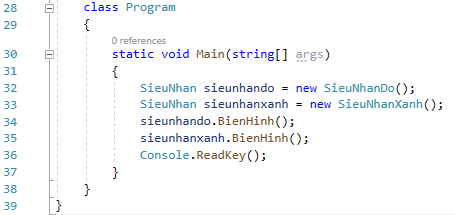
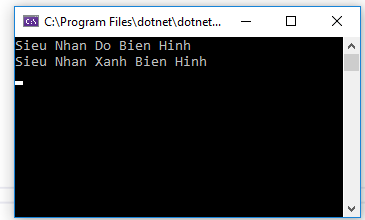
* **Polymorphism**
* **Definition: Tính đa hình**là các đối tượng thuộc các lớp khác nhau có thể hiểu cùng 1 thông điệp theo các cách khác nhau. Tính đa hình có 2 loại bao gồm overloading and overriding.
* **Overiding**

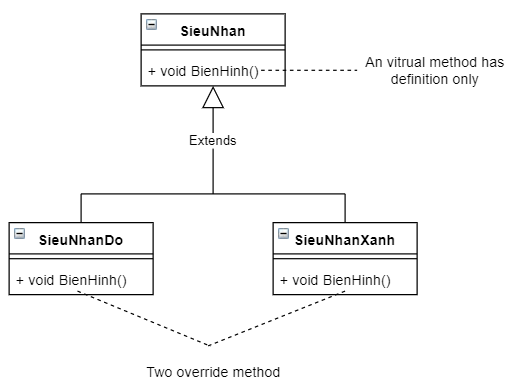




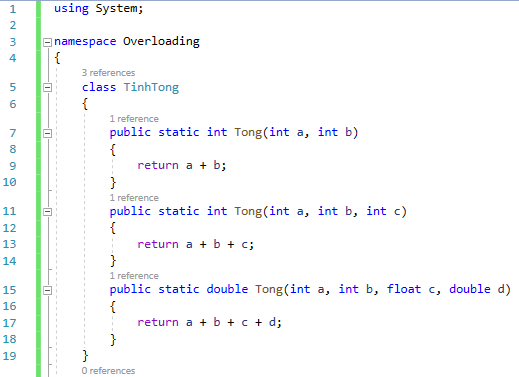
* **Explain:** Tôi tạo ra 3 lớp bao gồm SieuNhan, SieuNhanDo và SieuNhanXanh. Trong đó, 2 lớp SieuNhanDo và SieuNhanXanh kế thừa lớp SieuNhan. Bên trong class SieuNhan tôi tạo ra một phương thức BienHinh với phạm vi truy cập là virtual. Bên trong 2 class SieuNhanDo va SieuNhanXanh tôi cũng tạo ra 2 phương thức đều có tên là BienHinh với phạm vi truy cập là override. Override chỉ được sử dụng khi mà bạn đã khai báo lớp cha trước đó với phạm vi truy cập là virtual (ở đây có nghĩa là lớp cha SieuNhan với phạm vi truy cập virtual sẽ được lớp con SieuNhanDo và SieuNhanXanh kế thừa và 2 lớp con này có quyền được override lên phương thức BienHinh). Tính đa hình chỉ được thể hiện khi đã ghi đè lên phương thức của lớp cha.
* **Result:**

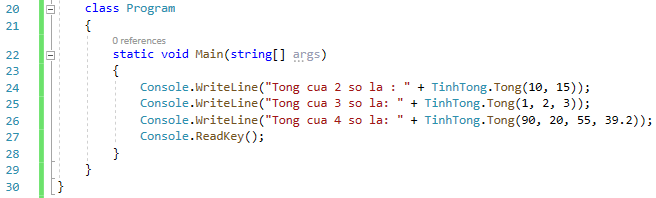


* **Class Diagram**

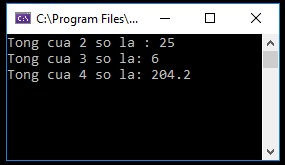


* Override method được sử dụng để thu được tính đa hình tại runtime.
* **Overloading**

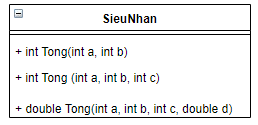




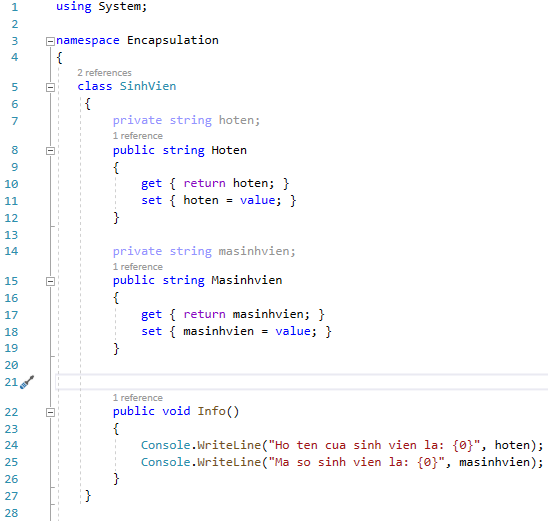
* **Explain:** As you can see Figure , tôi đã tạo ra một class có tên là TinhTong với 3 phương thức đều có tên giống nhau là Tong và đều thực hiện chung một phép toán tử đó là phép cộng nhưng có các tham số đầu vào khác nhau(khác nhau ở đây là phương thức thứ 3 có giá trị với kiểu dữ liệu double).
* **Result:**

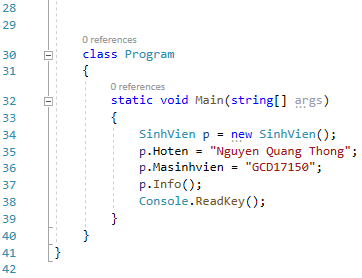


* **Class Diagram**

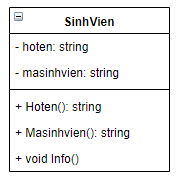
****

* Nạp chồng phương thức được sử dụng để thu được tinh đa hình tại compile time.
* **Encapsulation**
* **Definition: Tính đóng gói** là tính chất không cho phép người dùng hay đối tượng khác thay đổi dữ liệu thành viên của đối tượng nội tại. Chỉ có các hàm thành viên của đối tượng đó mới có quyền thay đổi trạng thái nội tại của nó mà thôi. Các đối tượng khác muốn thay đổi thuộc tính thành viên của đối tượng nội tại, thì chúng cần truyền thông điệp cho đối tượng, và việc quyết định thay đổi hay không vẫn do đối tượng nội tại quyết định.
* **Private Access Modifier**

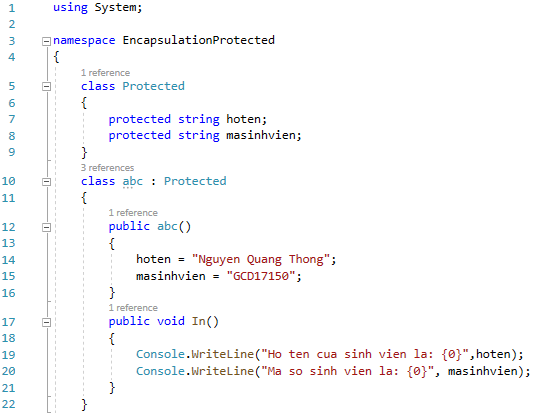


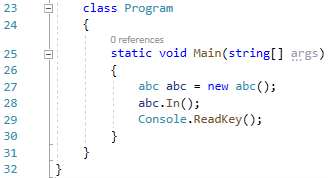


* **Explain:** As you can see Figure , tôi đã tạo một lớp có tên là SinhVien bao gồm 2 biến với phạm vi truy cập là private đều thuộc cùng kiểu dữ liệu string có tên là hoten và masinhvien. Nhưng để sử dụng được 2 giá trị này thì tôi bắt buộc phải tạo ra 2 biến khác có tên là Hoten và Masinhvien với phạm vi truy cập là public. Để có thể truy xuất được dữ liệu của 2 field Hoten và Masinhvien, tôi sử dụng property đó là get và set. Get sẽ trả về giá trị hoten và set sẽ gán giá trị bằng value. Tính đóng gói có ý nghĩa mang tính bảo mật cao tránh việc truy xuất dữ liệu một cách trái phép.
* **Class Diagram**

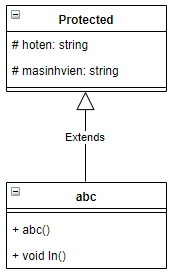


* **Protected Access Modifier**

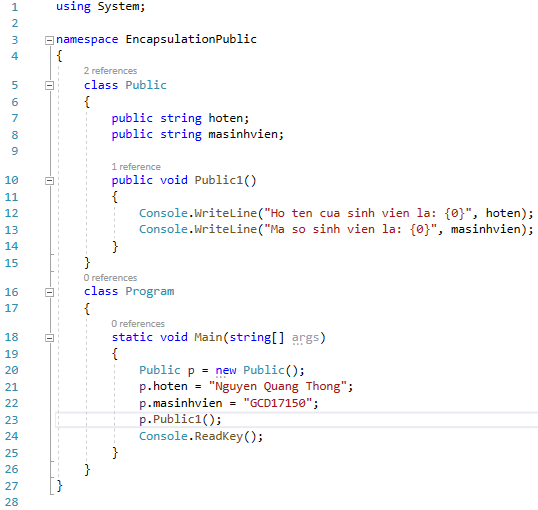




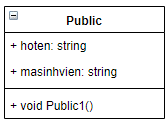
* **Explain:** Look at Figure , tôi đã tạo một lớp có tên là Protected bao gồm 2 biến với phạm vi truy cập là protected đều thuộc cùng kiểu dữ liệu string có tên là hoten và masinhvien. Nhưng để sử dụng được 2 giá trị này thì tôi bắt buộc phải tạo ra một class mới có tên là abc để kế thừa class Protected. Trong class abc tôi tạo ra một method abc để nhập dữ liệu từ 2 giá trị hoten và masinhvien và tôi cũng tạo ra method In để xuất dữ liệu ra.
* **Class Diagram**



* **Public Access Modifier**



* **Explain:** As you can see Figure , tôi đã tạo một lớp có tên là Public bao gồm 2 biến với phạm vi truy cập là public đều thuộc cùng kiểu dữ liệu string có tên là hoten và masinhvien và tôi cũng tạo ra trong class Public một phương thức có tên là Public1 để xuất giá trị ra. Với public access modifier thì việc sử dụng lại 2 tên biến hoten và masinhvien là cực kì dễ dàng, public access modifier có phạm vi truy cập rộng nhất so với hai modifier trên mà tôi đã trình bày.
* **Class Diagram**

****

* **Result of 3 access modifiers in Encapsulation:**

