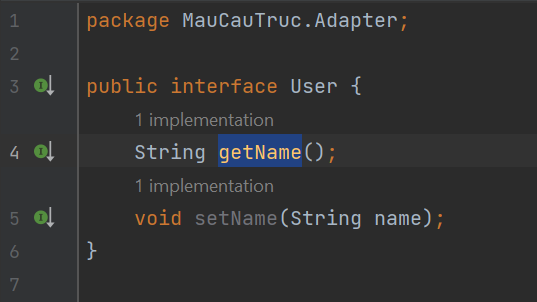
**BÁO CÁO MẪU THIẾT KẾ**

1. **Mẫu cấu trúc**
   1. **Adapter**
2. **Bài toán**

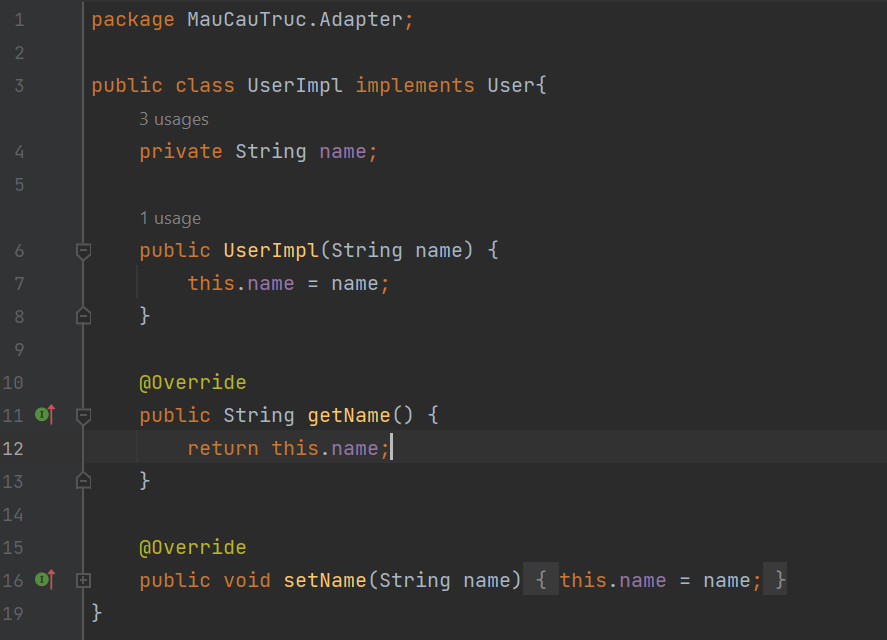
* Có một hệ thống cũ sử dụng họ và tên nhưng hệ thông mới lại muốn tách họ và tên riêng

1. **Cài đặt chương trình**

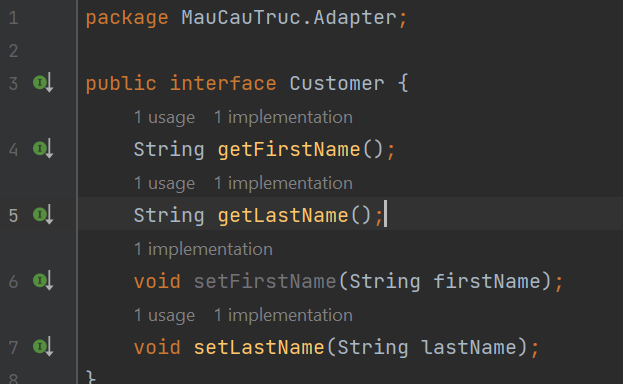
* Ta có interface User chứa 2 phương thức getName() và setName()



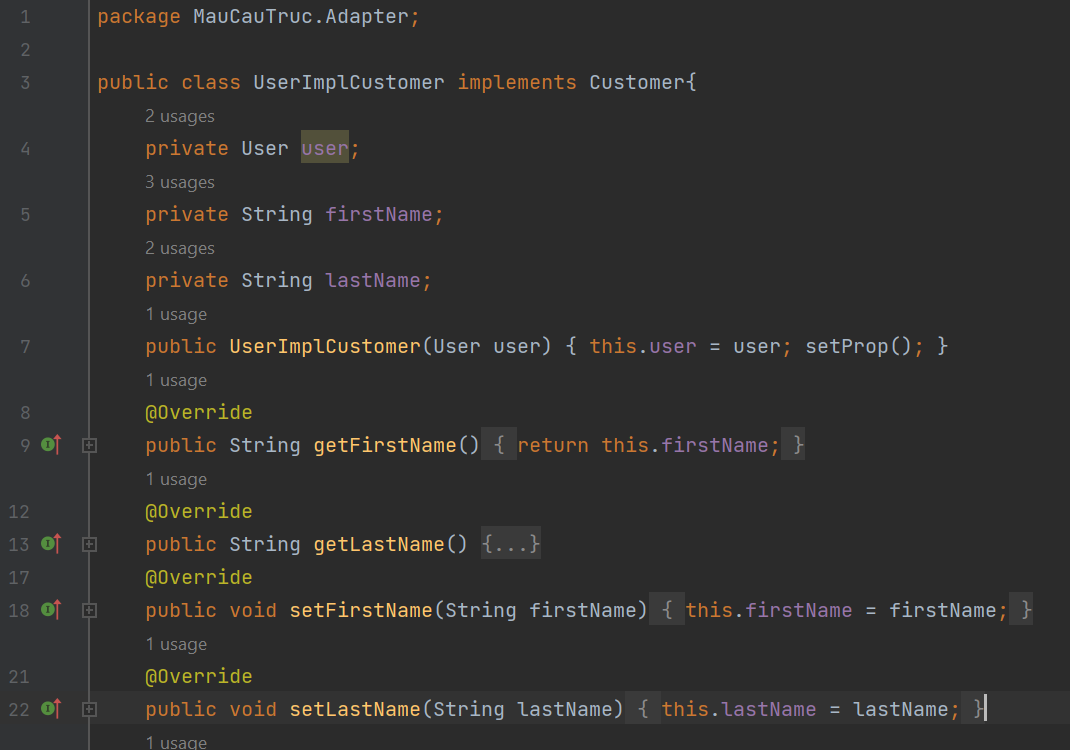
* Lớp UserImpl được triển khai từ interface User. Trong đó có duy nhất một thuộc tính là name để lưu trữ họ và tên.

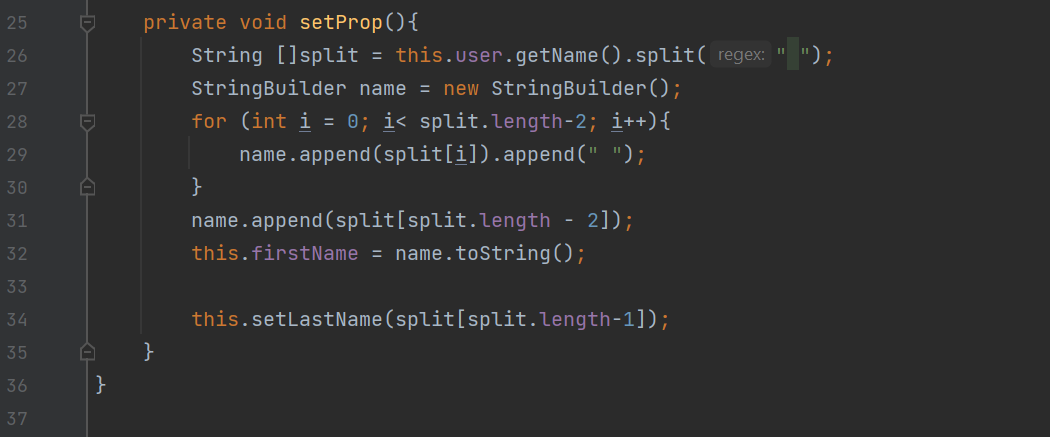


* Một interface mới Customer chứa 2 method getFirstName(), getLastName(), setFirstName(), setLastName() đại diện cho hệ thống mới cần được phát triển từ hệ thống cũ.



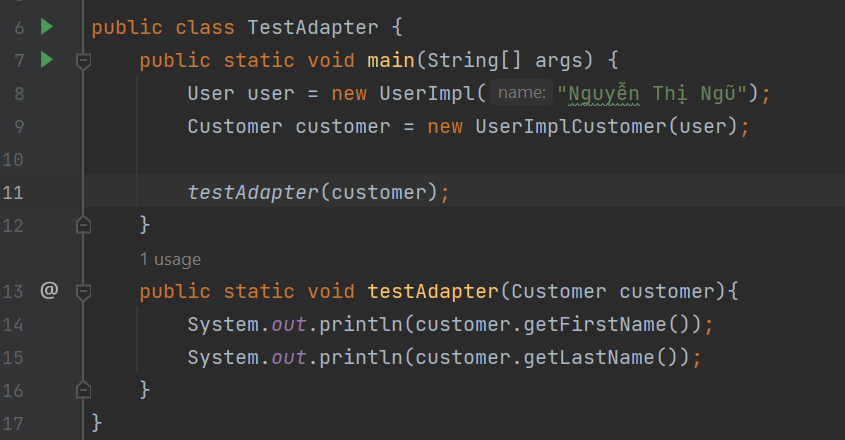
* Hệ thống mới cần giữ thông tin của một User, từ đó tách name của user thành 2 phần firstName và lastName. Method setProp() có tác dụng tách thông tin cũ thành thông tin mới để lưu trữ.



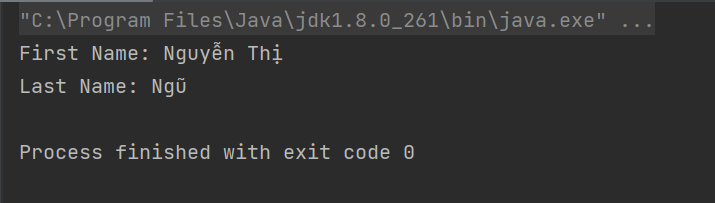


1. **Kết quả**

* Viết hàm main để test. Tạo ra đối tượng User có name là Nguyễn Thị Ngũ. Đối tượng Customer được tạo từ User và tách được firstName và lastName.



* Kết quả:



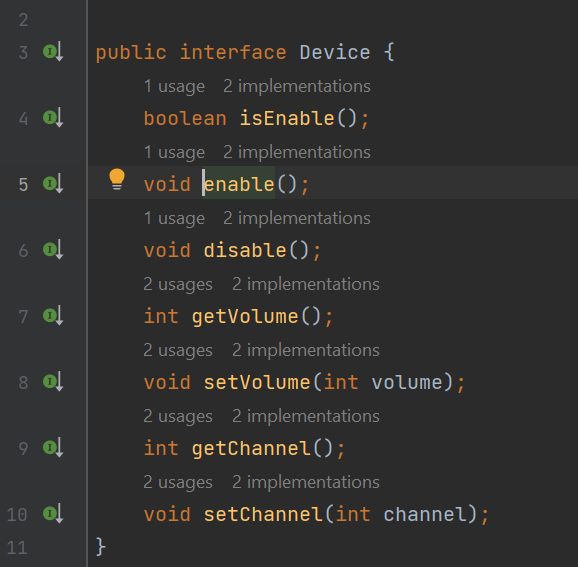
* 1. **Bridge**

1. **Bài toán**

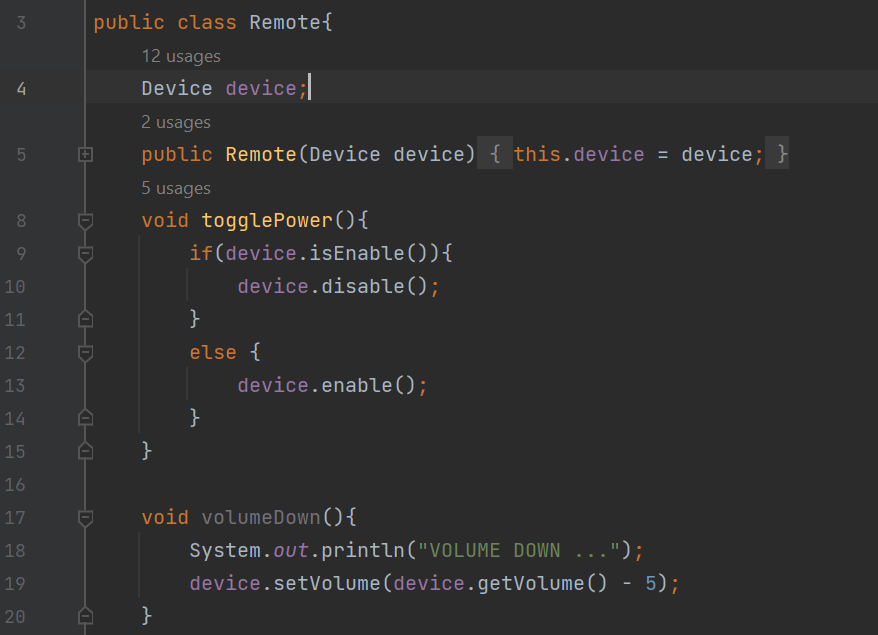
* Tạo ra hệ thống điều khiển từ xa. Tất cả các thiết bị điều khiển từ xa có một số tính năng giống nhau.

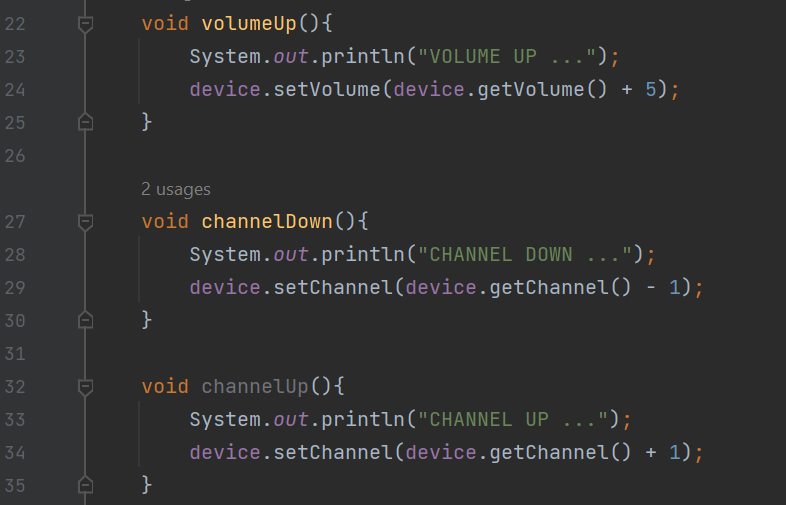
1. **Cài đặt chương trình**

* Tạo một interface Device có các method chung

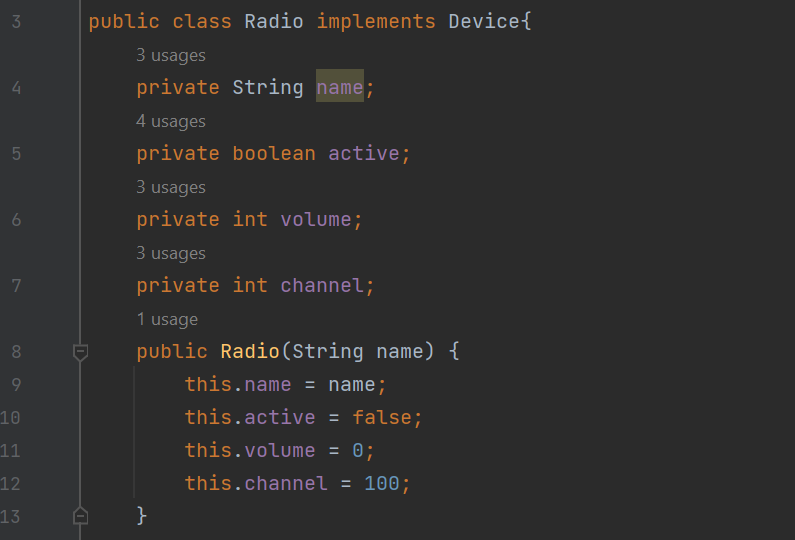


* Tạo một lớp Remote cài đặt các tính năng chung của các thiết bị

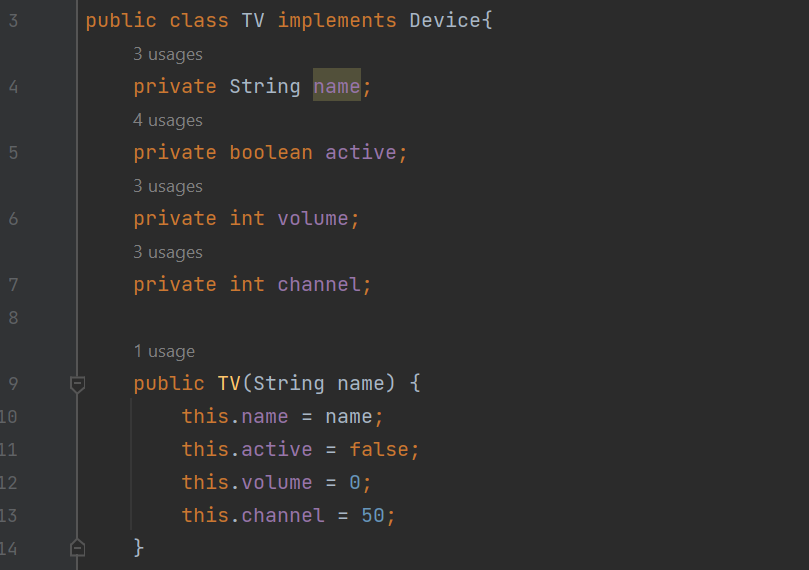




* Tạo ra các lớp con từ interface trên. Các method cần được ghi đè lại tùy thuộc vào mỗi loại thiết bị.



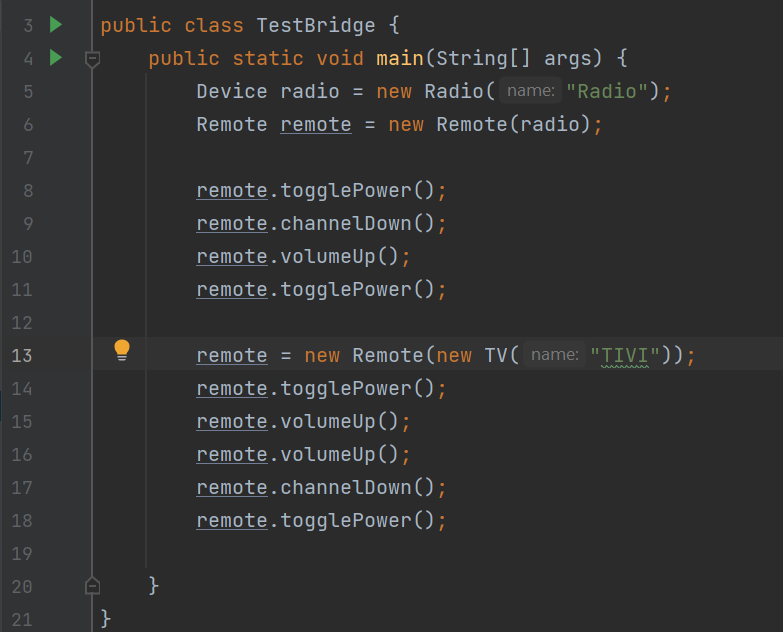




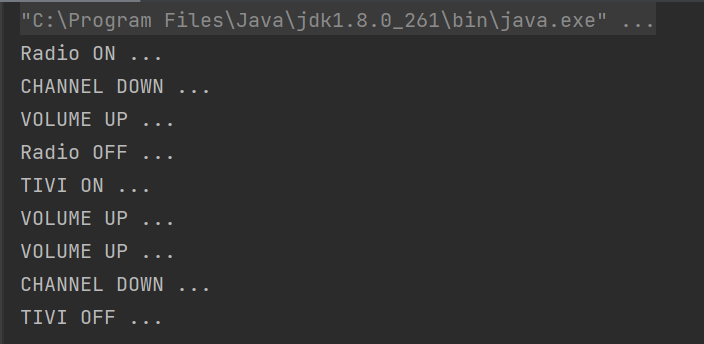


1. **Kết quả**

* Tạo main để test. Bên trong ta tạo một đối tượng Radio và gán nó cho giao diện Device.
* Tiếp đến tạo một đối tượng Remote và chuyền đối tượng vừa tạo ở phía trên vào. Gọi method togglePown() để bật/tắt thiết bị.
* Gọi method channelUp()/channelDown() để tăng/giảm kênh.
* Gọi method volumeUp()/volumeDown() để tăng giảm âm lượng.



* Kết quả chạy:



* 1. **Composite**

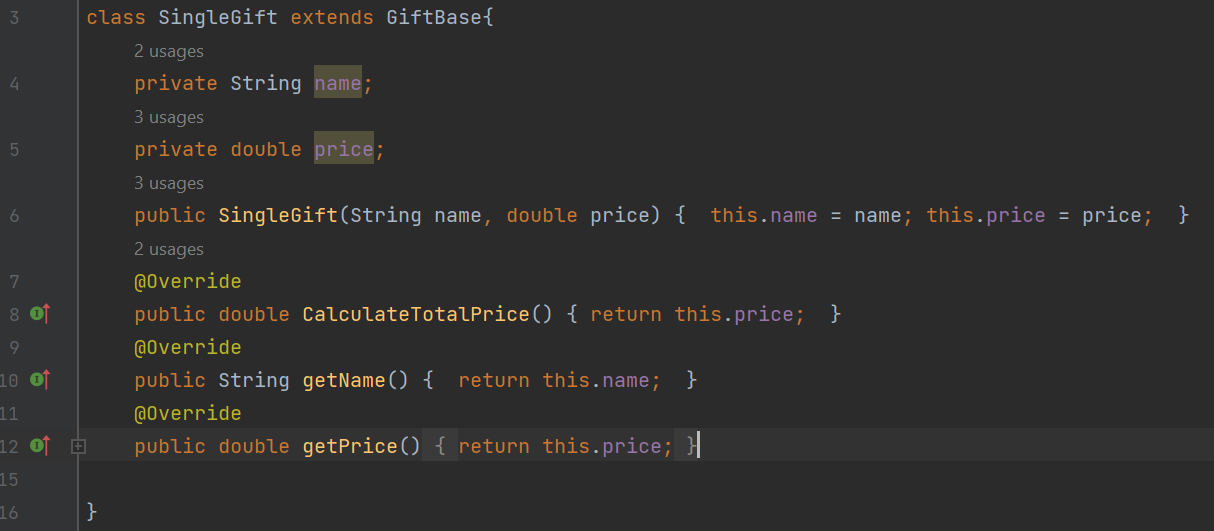
1. **Bài toán**

* Trong một cửa hàng có những hộp quà khác nhau. Bên trong hộp quà có thể là một món đồ hoặc một hộp quà khác. Mỗi món đồ có một giá trị khác nhau. Người chủ muốn tính xem hộp quà này có giá trị bao nhiêu.

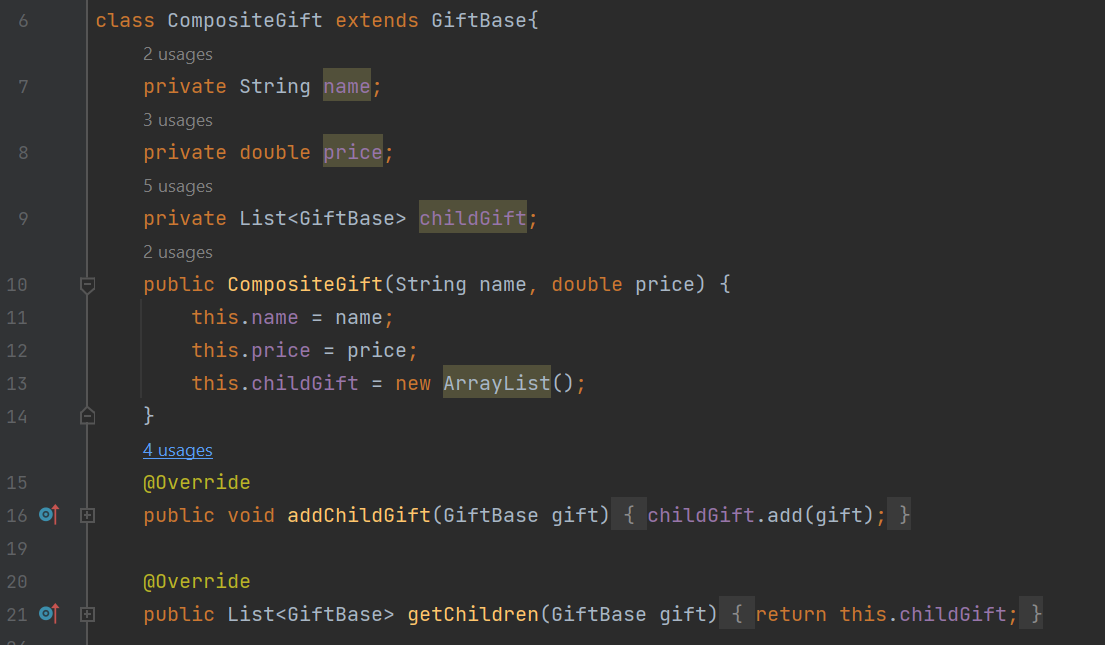
1. **Cài đặt chương trình**

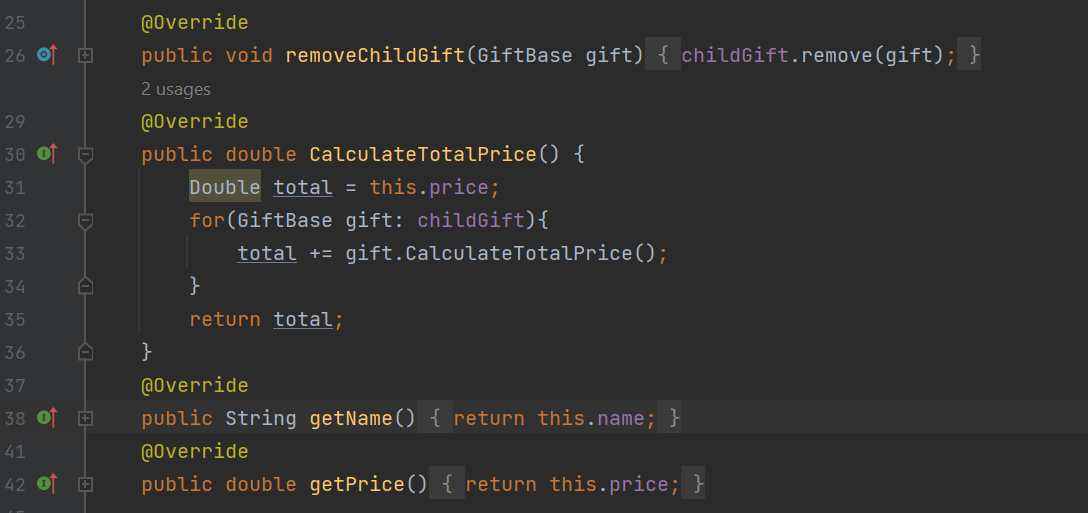
* Lớp GiftBase để có 2 methohd abstract là getName() để lấy tên hộp quà và getPrice() để lấy giá trị. Lớp này còn có method addChildGift(), removeChildGift(), getChildren(), để thêm/bớt và lấy ra các hộp quà con bên trong. Một hàm CalculateTotalPrice() để tính tổng giá trị quà.

- Lớp SingleGift thể hiện là một món quà. Được kế thừa từ lớp GiftBase, chưa hai thuộc tính name và price. Ta ghi đè lại các method abstract từ lớp cha.



* Tạo một lớp CompositeGift có các thuộc tính name, price và list<GiftBase>



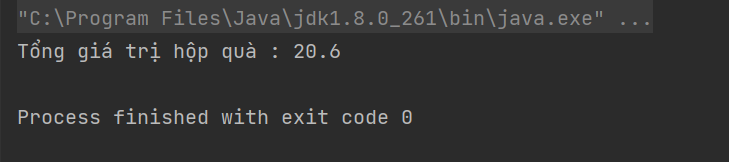


1. **Kết quả**

* Tạo main để test chương trình. Tạo ra 2 hộp quà và 3 món quà. Thêm vào hộp quà 1 một vài món và sau đó cho hộp quà 1 và hộp quà 2 và 1 món quà nữa. Tính tổng giá trị hộp quà.



* Kết quả:

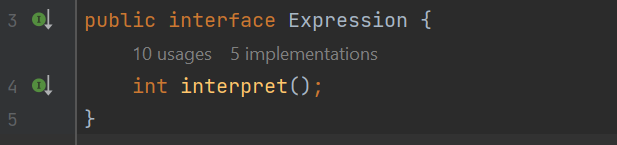


1. **Mẫu hành vi**
   1. **Interpreter**
2. **Bài toán**

* Cho biểu thức tiền tố. Ta cần tính giá trị của biểu thức

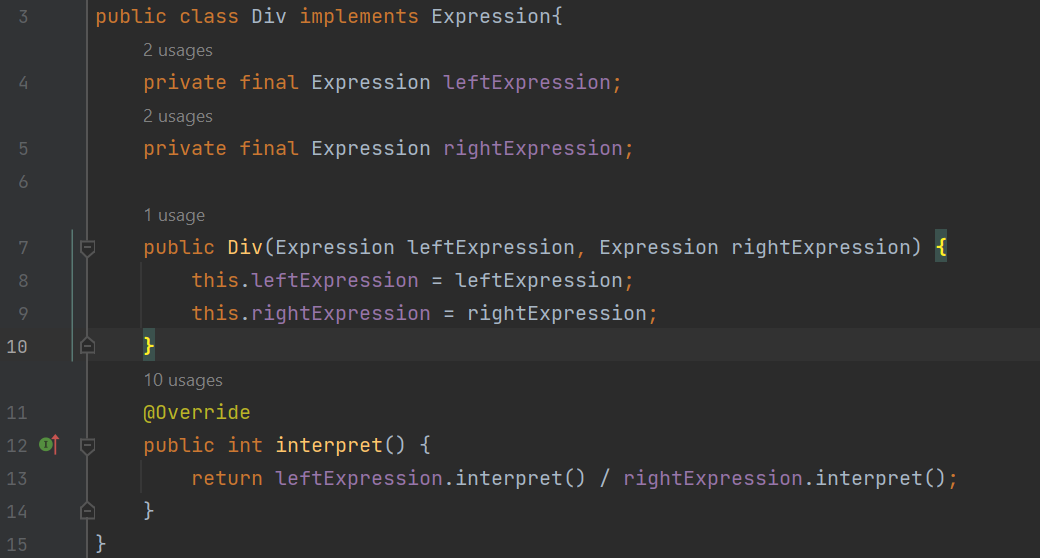
1. **Cài đặt chương trình**

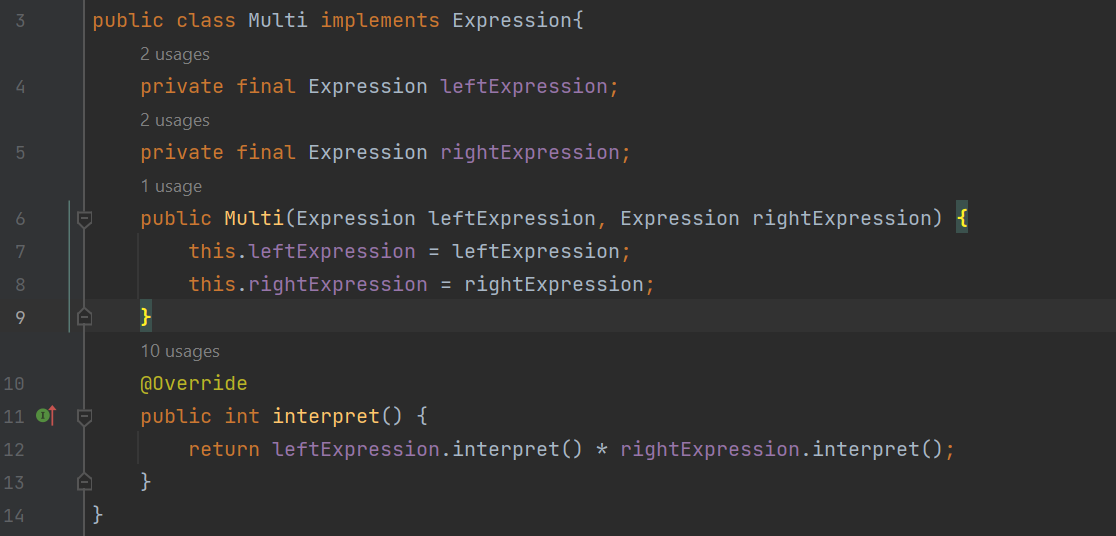
* Tạo một interface Expression để lưu hành động.

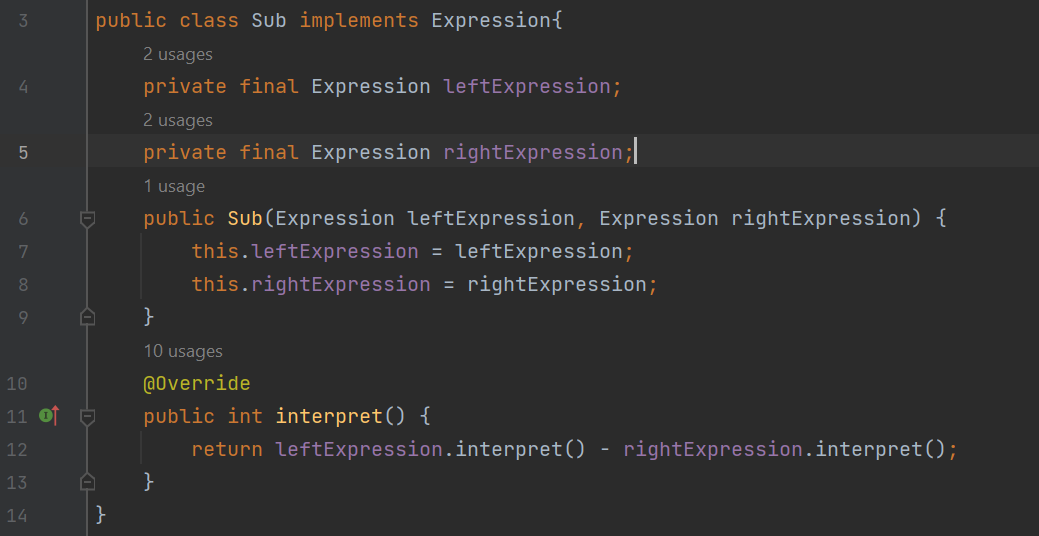


* Tạo ra class là các phép toán. Các class này kế thừa từ Expression. Bên trong chứa 2 thuộc tính là biểu thức trái và biểu thức phải. Một phương thức được override lại để tính toán.

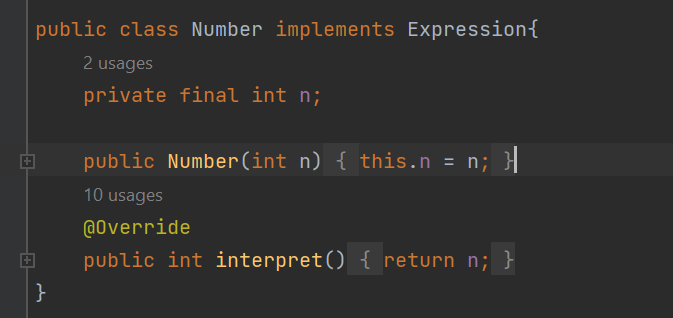




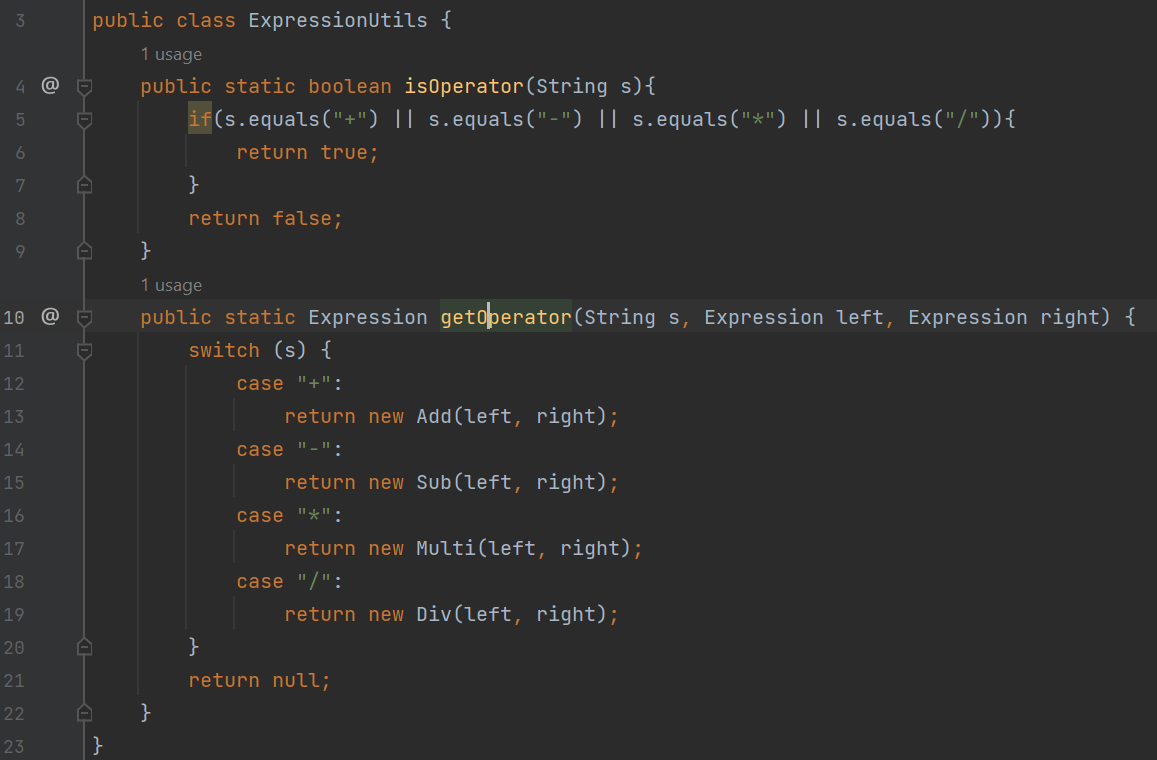




* Một lớp Number để lưu số.

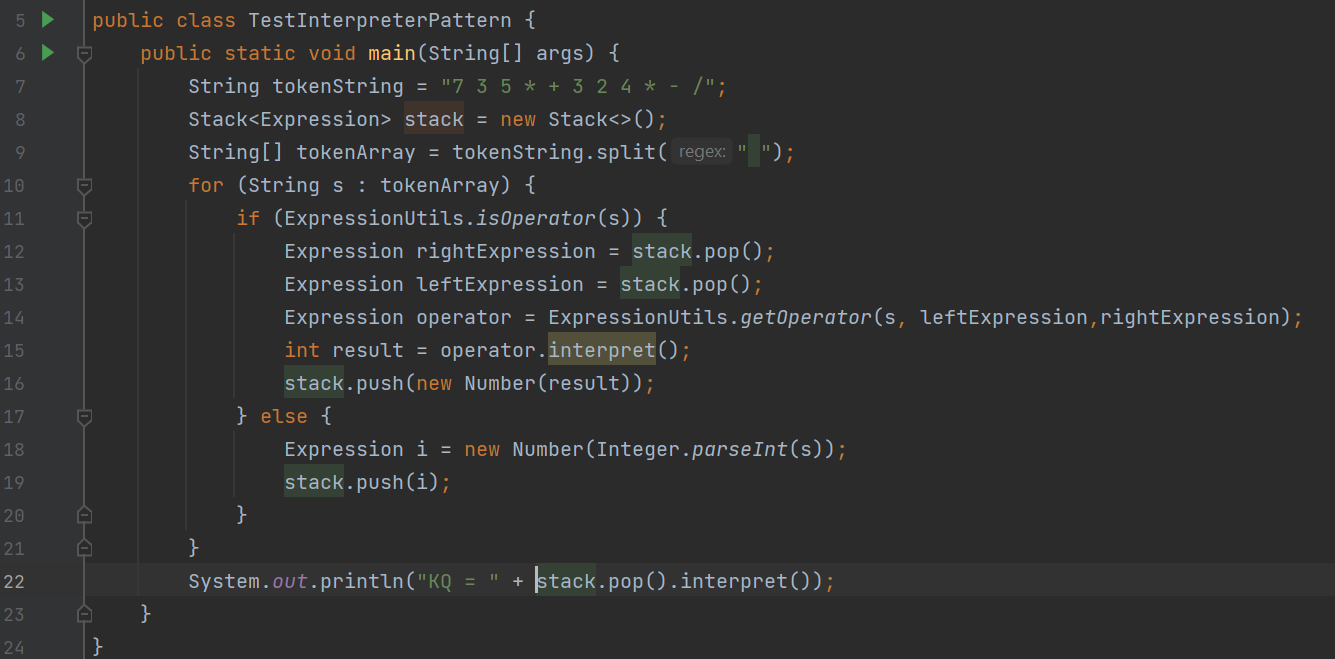


* Một lớp ExpressionUtils có 2 method, 1 method để kiểm tra xem chuỗi truyền vào là số hay biểu thức. 1 method để tính toán, ta cần truyền 2 biểu thức và một phép toán.

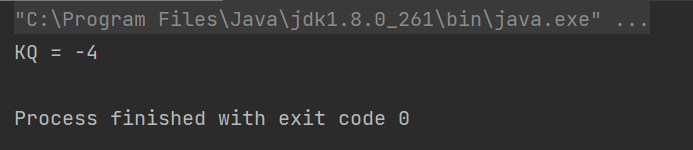


1. **Kết quả**

* Viết phương thức main để test. Trong main ta có một biểu thức tiền tố, 1 ngăn xếp để tính toán. Lấy từng chuỗi thêm vào ngăn xếp. Nếu chuỗi là số thì ta thêm trực tiếp và ngăn xếp, còn nếu chuỗi là phép tính thì ta lấy 2 số đầu tiên của ngăn xếp rồi tính toán. Sau đó thêm kết quả vào ngăn xếp. Kết quả cần tìm chính là phần tử còn lại duy nhất của ngăn xếp.



* Kết quả:



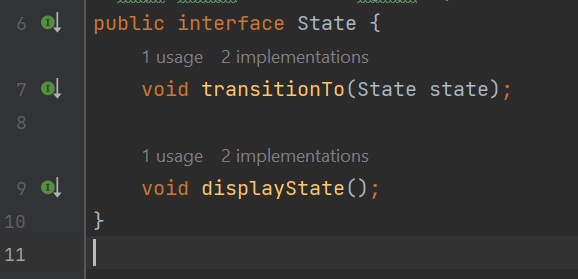
* 1. **State**

1. **Bài toán**

* Người dùng muốn thay đổi trạng thái của chiếc điện thoại. Khi điện thoại tắt, người dùng ấn nút nguồn điện thoại sẽ mở và ngược lại.

1. **Cài đặt chương trình**

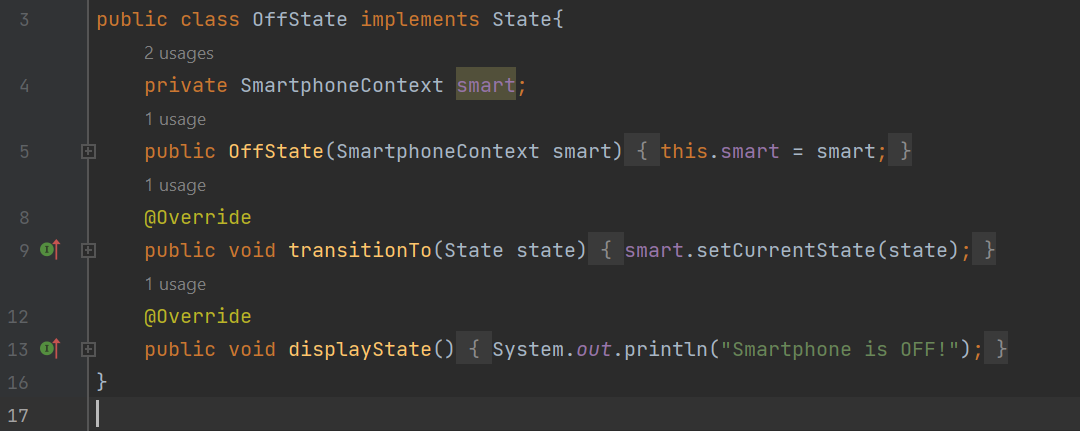
* Có 1 interface chứa 2 method là hiển thị và chuyển đổi

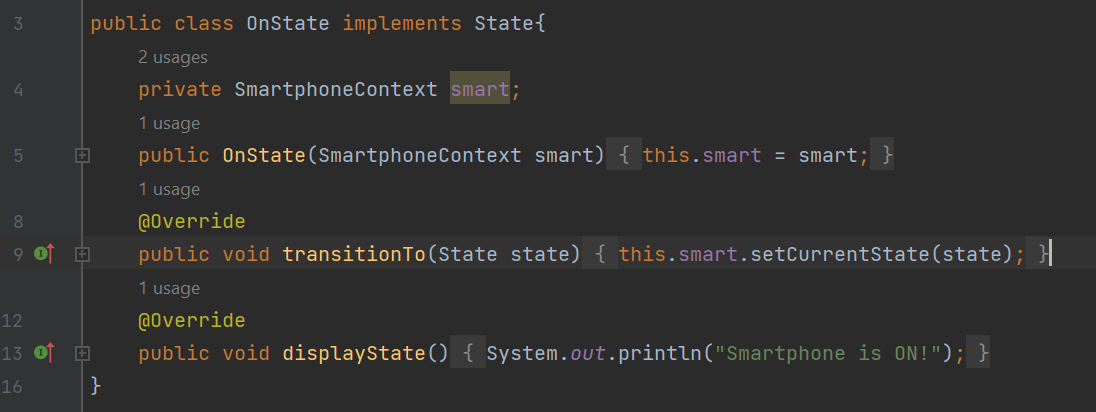


* Một lớp SmartphoneContext chứa đối tượng hiện thời.



* Tiếp đến ta tạo 2 class được triển khai từ interface trên. Bên trong mỗi class lưu trữ 1 đối tượng SmartphoneContext và thay đổi trạng thái của đối tượng.



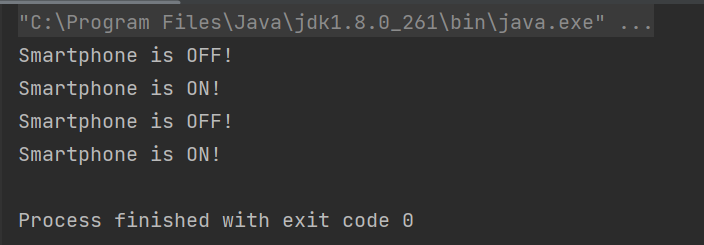


1. **Kết qủa**

* Tạo method main để test chương trình. Tạo một đối tượng smartphone, hiển thị trạng thái hiện thời, sau đó thay đổi trạng thái của máy và hiển thị lại



* Kết quả:



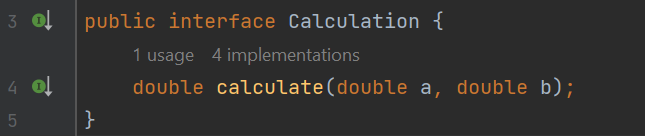
* 1. **Strategy**

1. **Bài toán**

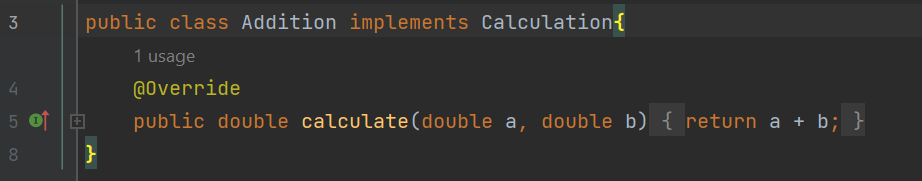
* Có các phép tính khác nhau, dựa vào phép tính để tính ra kết quả

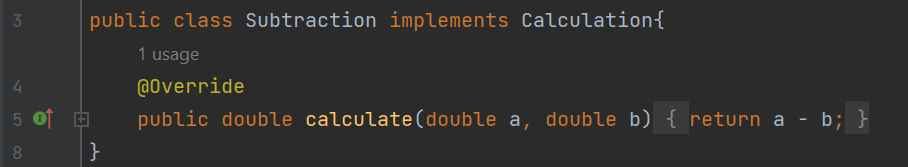
1. **Cài đặt chương trình**

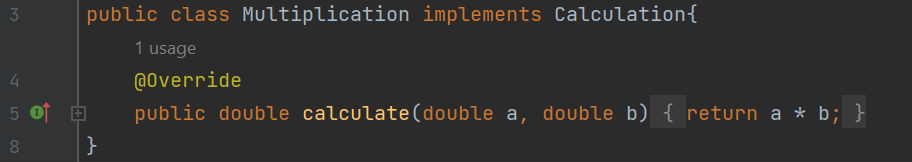
* Tạo một interface Calculation có method là calculate() chuyền vào 2 số nguyên.

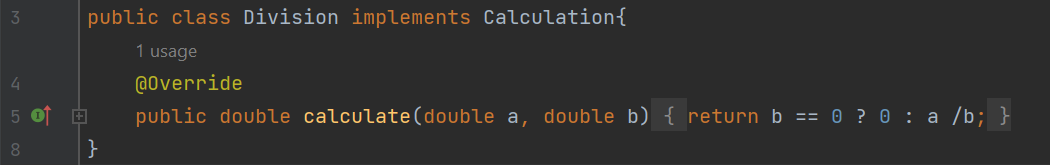


* Tạo các class được triển khai từ interface trên. Các class là các phép toán khác nhau.

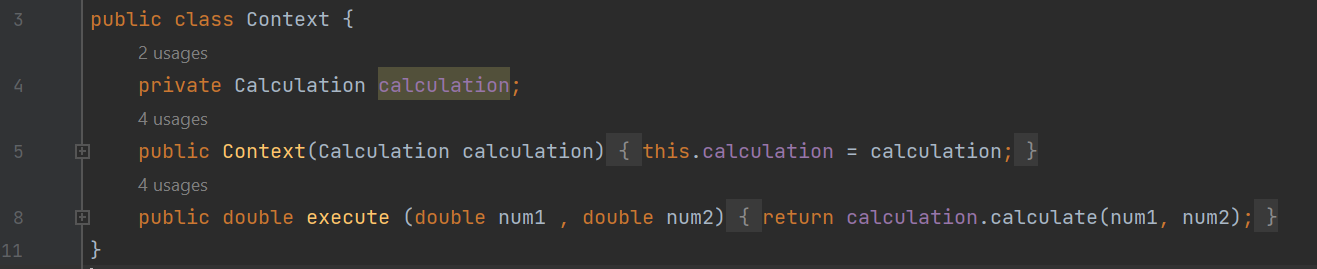






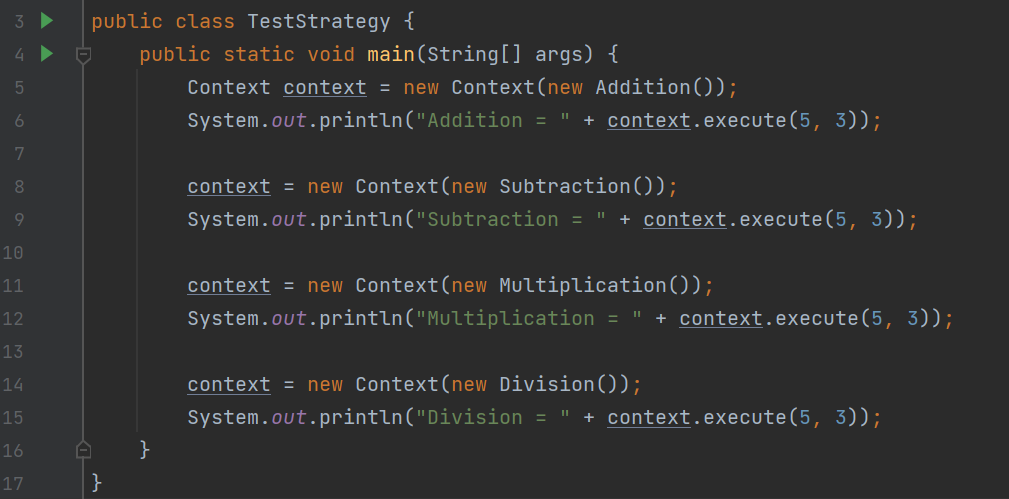


* Tạo class chứa phép toán hiện thời. Một method để thực thi phép toán đó.

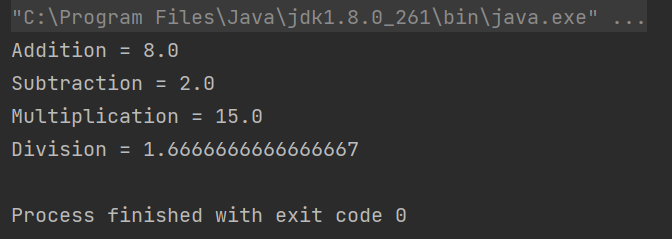


1. **Kết quả**

* Tạo method main để test



* Kết quả:

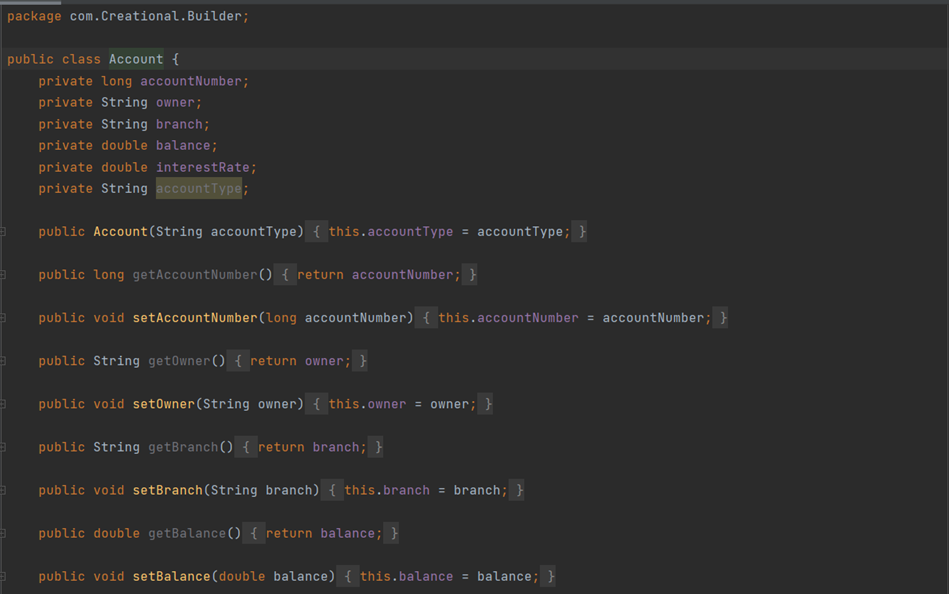


1. **Mẫu khởi tạo**
   1. **Builder**
2. **Bài toán**

* Trợ giúp ngân hàng đưa ra thông tin các loại tài khoản khác nhau của người dùng hiển thị ra ứng dụng, yêu cầu cung cấp đối tượng Account, đối tượng này cần chứa đặc tả của Account. Ứng dụng sẽ sử dụng đặc tả để hiển thị thông tin tài khoản. Có các loại tài khoản khác nhau như BankAccount và SavingAccount. Chỉ có một đối tượng Account và chúng ta sẽ tạo đối tượng Account tuân theo phân lớp đó. Hiện tại chỉ có hai loại tài khoản, sau này có thể có thêm các loại tài khoản khác.

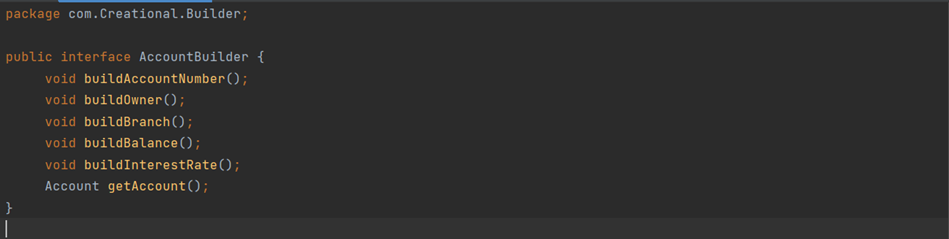
1. **Cài đặt chương trình**

* Dưới đây là lớp Account mà chứa một số thành phần quan trọng của tài khoản mà được yêu cầu để xây dựng đối tượng Account hoàn chỉnh.

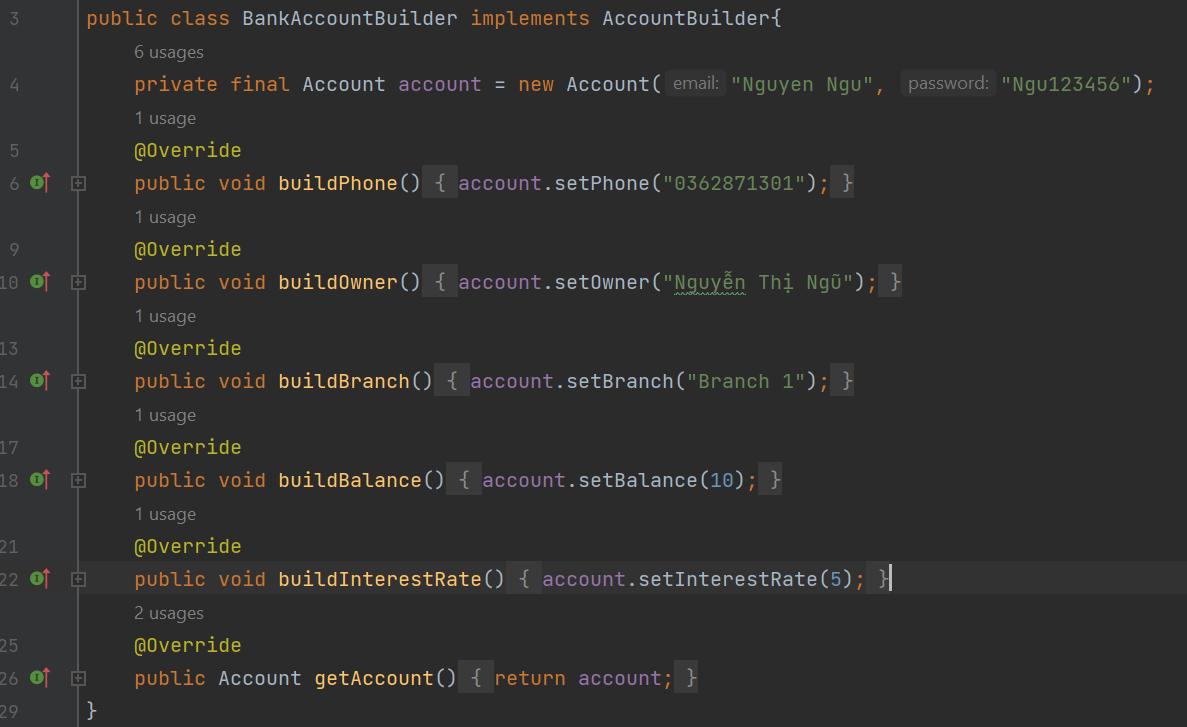


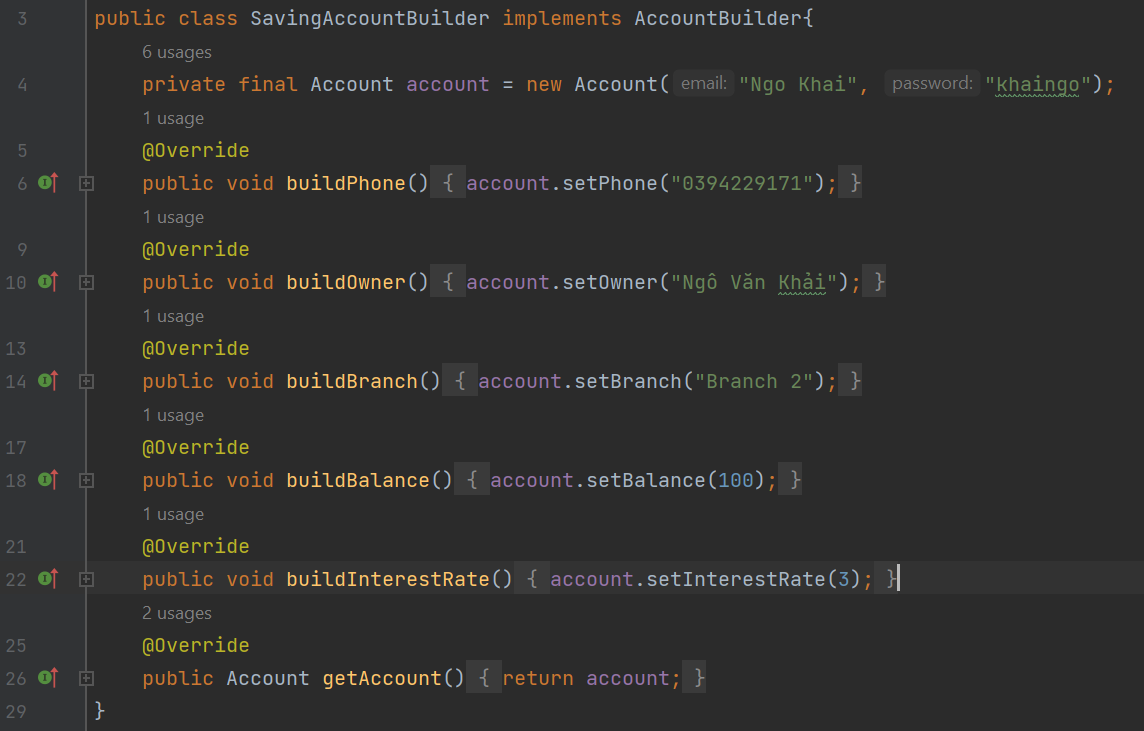


* AccountBuilder là giao diện Builder chứa tập các phương thức được sử dụng để xây dựng đối tượng account và các thành phần của nó.Phương thức getAccount được sử dụng để trả về đối tượng account cuối cùng sau khi tạo.

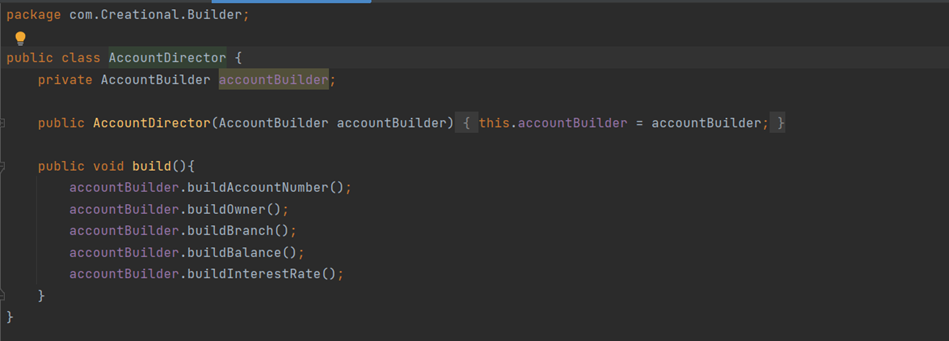


* Dưới đây là hai cài đặt của giao diện AccountBuilder, mỗi loại tương ứng với một loại Account, là BankAccount và SavingAccount.



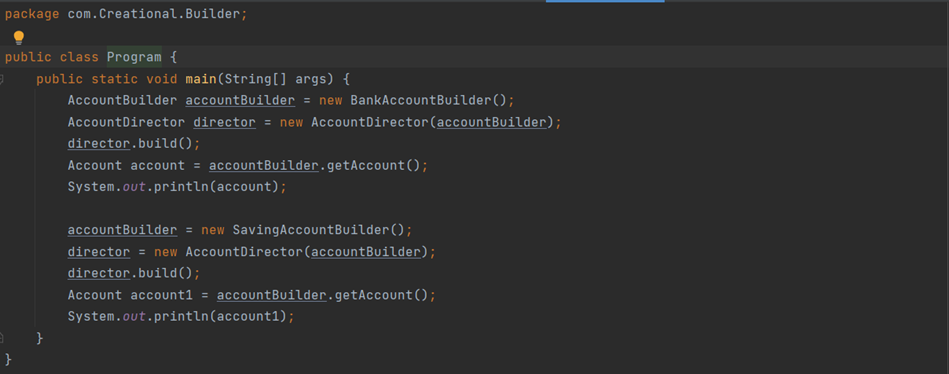


* Đây là lớp Director, lưu giữ các bước tương ứng tạo nên các thành phần để tạo ra sản phẩm account.

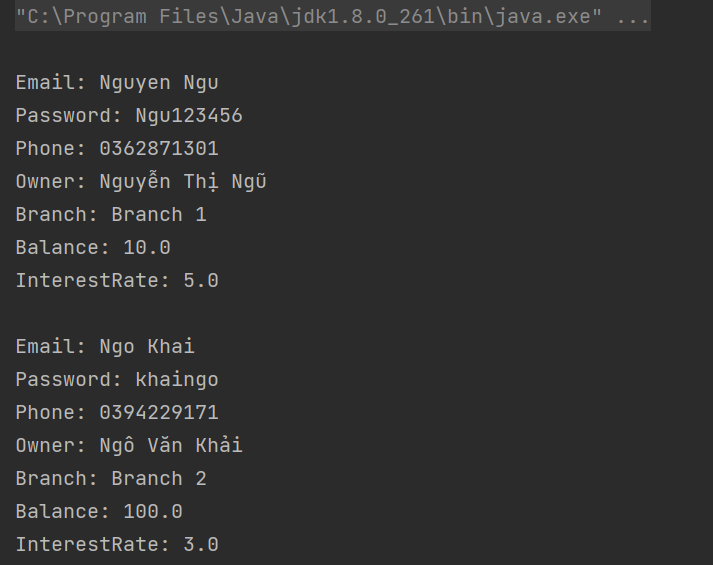


1. **Kết quả**

* Viết method main để kiểm thử



* Kết quả:



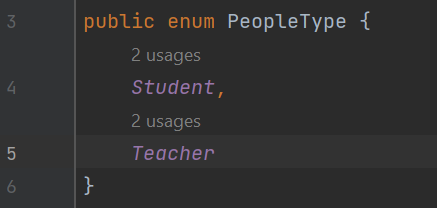
* 1. **Factory Method**

1. **Bài toán**

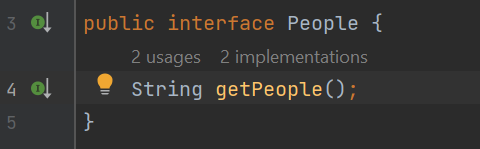
* Có nhiều đối tượng khác nhau, ta muốn tạo các đối tượng mới dựa vào kiểu đối tượng mong muốn.

1. **Cài đặt chương trình**

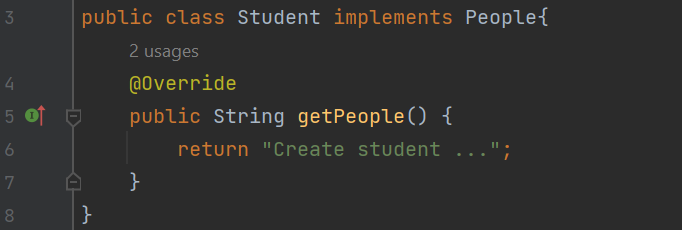
* Có một đối tượng enum để lưu trữ các kiểu đối tượng mong muốn tạo.

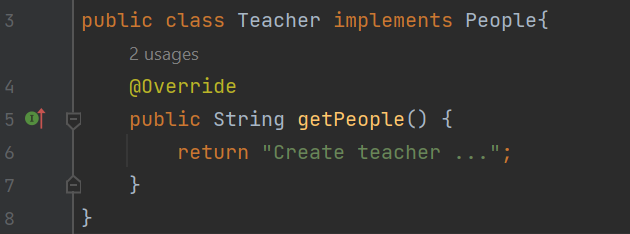


* Một interface có method để lấy ra đối tượng.

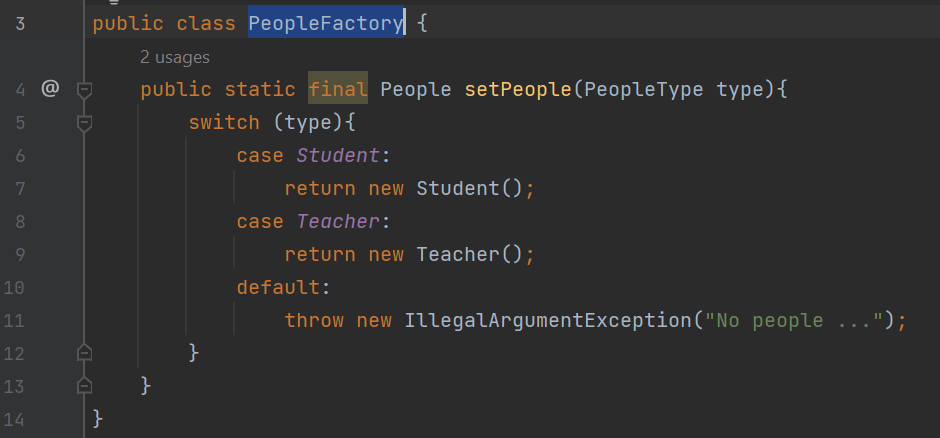


* Tạo các class là các đối tượng cần thiết.



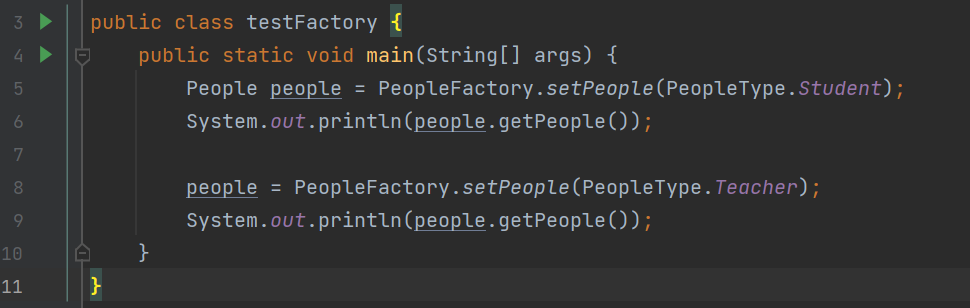


* Tạo ra class PeopleFactory để khởi tạo đối tượng cần thiết.

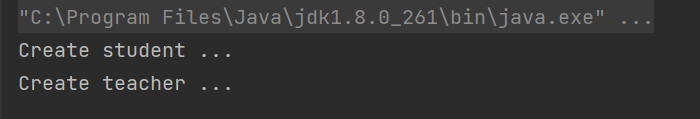


1. **Kết quả**

* Viết method main để kiểm thử



* Kết quả:



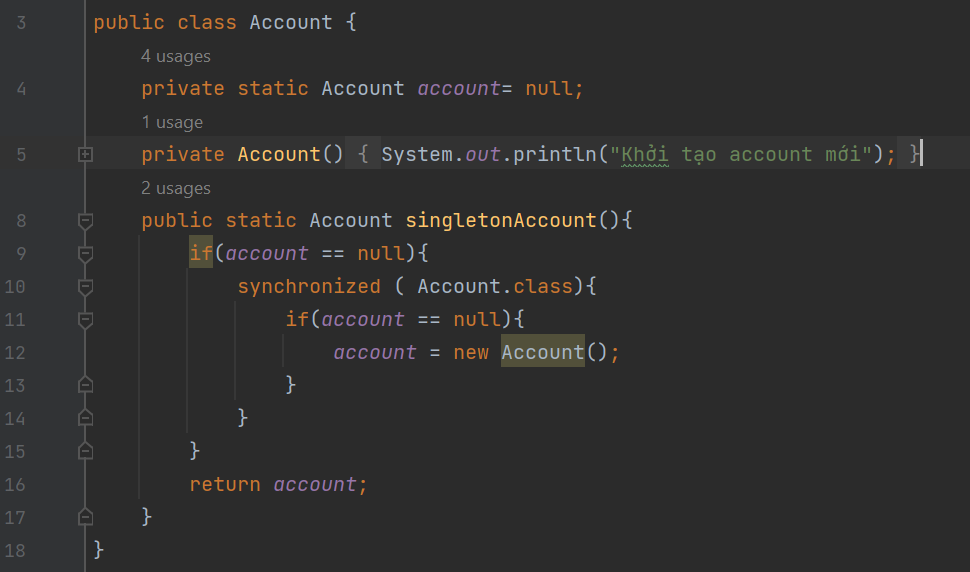
* 1. **Singleton**

1. **Bài toán**

* Khi muốn chỉ tạo duy nhất một đối tượng xuyên suốt quá trình chạy

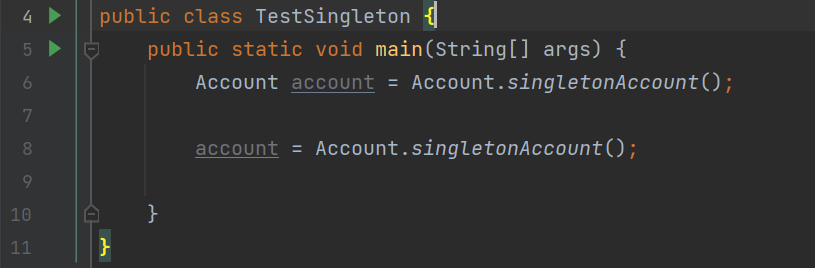
1. **Cài đặt chương trình**

* Trước khi tạo đối tượng ta kiểm tra xem có đối tượng hay chưa, nếu chưa thì tạo nếu có rồi thì dùng luôn. Để phục vụ đa luồng ta thêm từ khóa ‘synchronized’ để tạo hàng chờ cho các luồng.



1. **Kết quả**

* Viết chương trình main để kiểm tra



* Kết quả:

