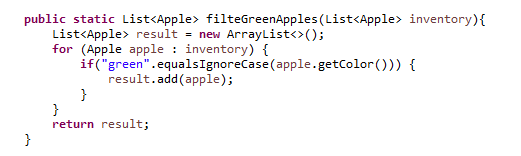
1. Passing code with behaviorparameterization:

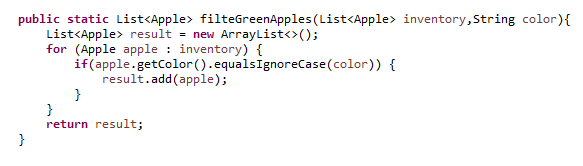
* Người nông dân bảo bạn lựa ra trái táo màu Xanh.

Bạn ngay lập tức viết :

****

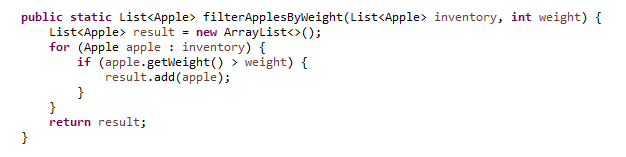
* Sau đó người nông dân update thành màu đỏ.

Bạn sửa lại thành:

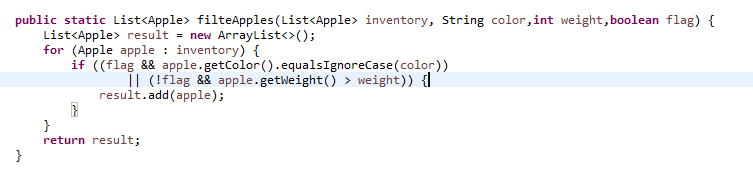


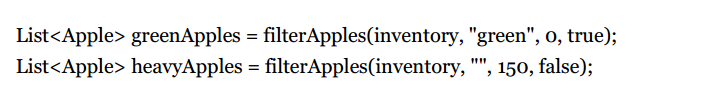
* Đây chính là bạn đã - **parameterizing the color**
* Nông dân sau khi lọc theo màu muốn thêm cả lọc theo cân nặng > 150g.

Cách dễ nhất là tạo ra method mới:



Nhưng bạn phải áp dụng 2 lần filter cho 1 apple.

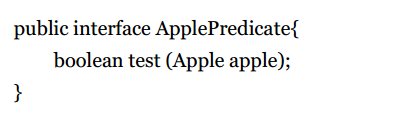
Để cải thiện nó, bạn viết thành như sau :

Sau đó chương trình chạy:

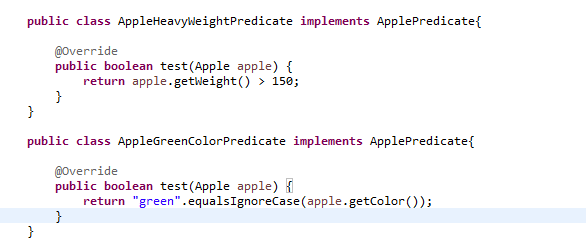
* Code thật khó nhìn, không thể đáp ứng đc nếu a Nông Dân thêm filter theo Size, Type,…
* Nên dùng **behavior parameterization**

1. **Predicate**

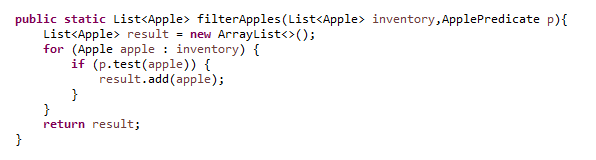
* Bạn đang làm việc với đống Apples và muốn trả về một Boolean Based trên một vài Attributes của Apple.(color, weight,size,…)
* Define an interface to model the selection criteria:



* Tạo 2 strategies(behaviors):



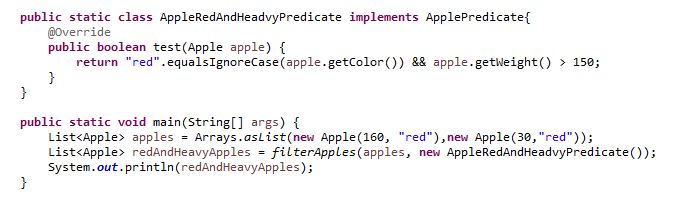
* Bạn muốn filterApples method chấp nhận ApplePredicate để kiểm tra một Apple.



* Behavior parameterization có nghĩa là:

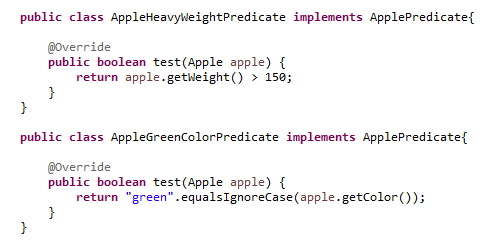
Khả năng nói với 1 method lấy nhiều behaviors (or Strategies) như các parameters và sử dụng chúng nội bộ để thực hiện các hành vi khác nhau.

Nếu ng nông dân muốn bạn lấy trái màu đỏ và > 150g:



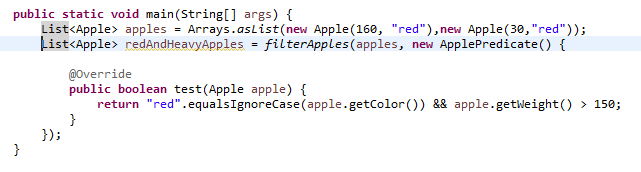
* Bạn đã parameterized behavior of filterApples method.

Có thể nhận thấy rằng chúng ta đã phải tạo ra khá nhiều class:

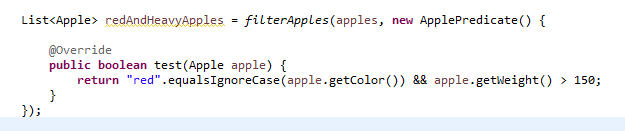


Java có hỗ trợ chúng ta declare và instantiate class cùng một thời điểm.

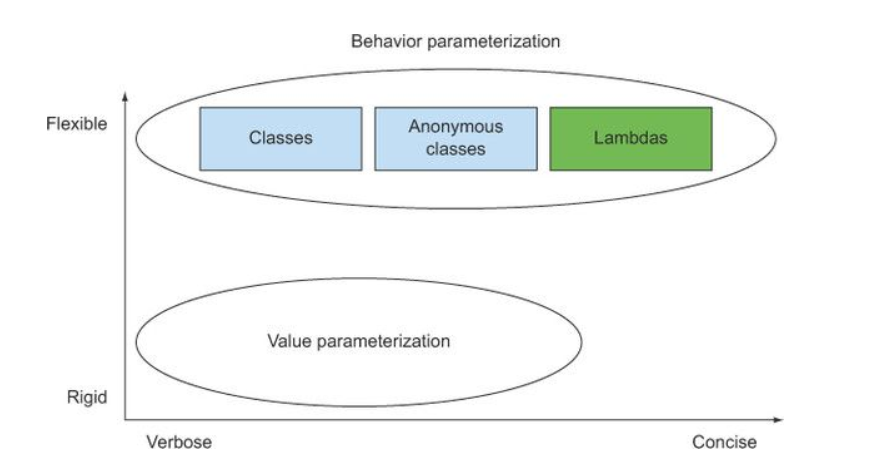
1. **Anonymous classes:**

Java có hỗ trợ chúng ta declare và instantiate class cùng một thời điểm.

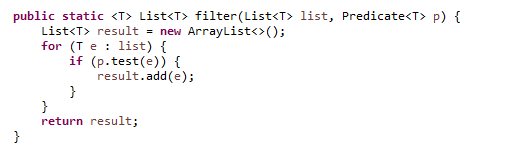
* Chúng ta thấy rằng anonymous class với thân hàm khá cồng kềnh, khó đọc và tách biệt. Vì thế, có Lamda Expression:



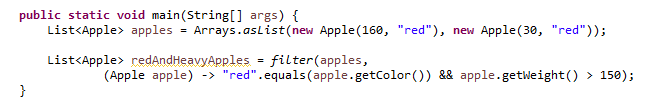
D:\Projects\ENEL_ENERGIA_HEROKU\TASKS\11-8\Images\11.PNG



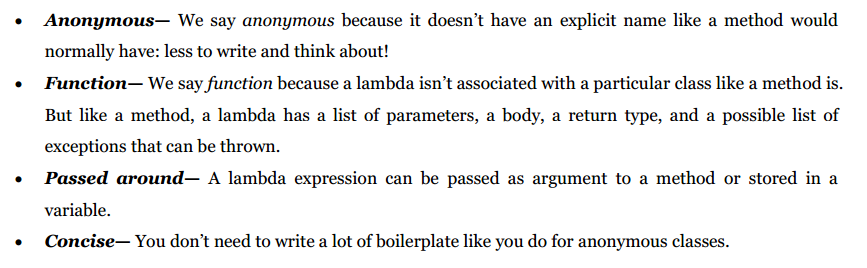
1. Abstracting List Type:

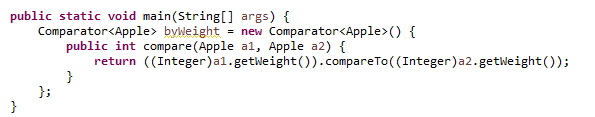


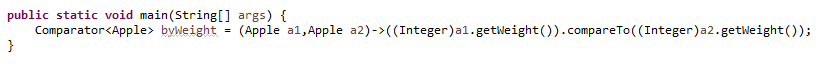
Và sử dụng nó :



1. Lambda expression:







1. A lambda expression is composed of parameters,an arrows and a body:



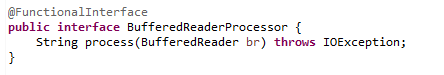
* Using {} to Lambda body :

() -> {return “IFI”;}

* Using without {} in lambda body:

() -> “IFI”

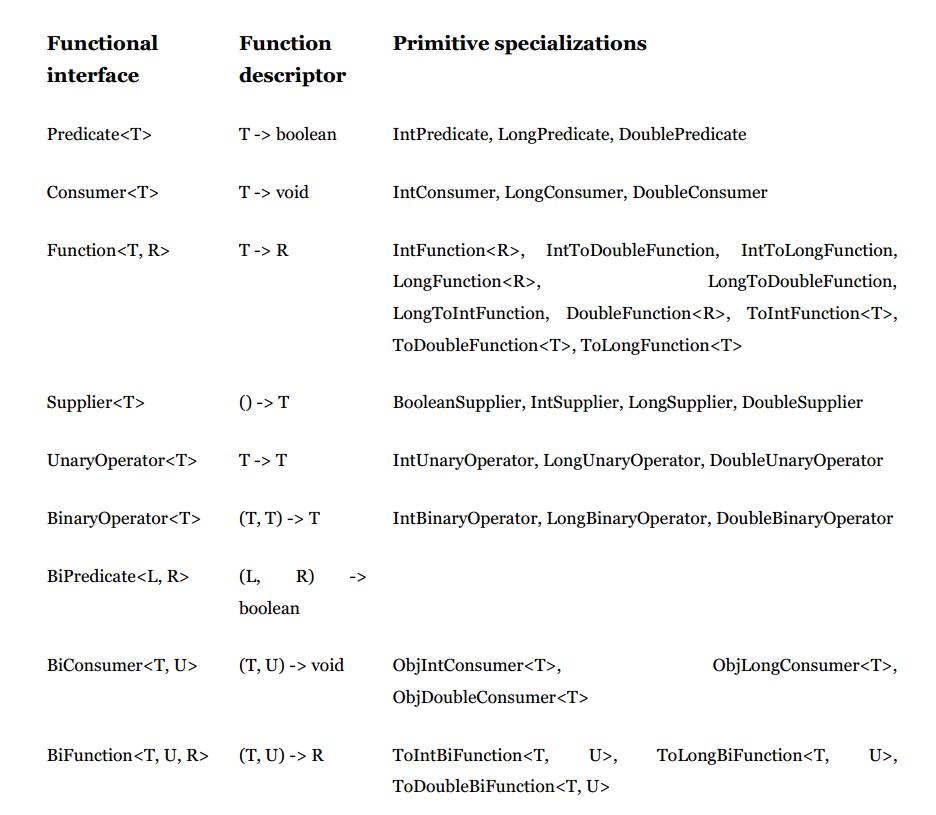
1. Functional Interface:



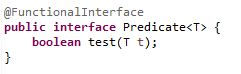
* Functional Interface chỉ có duy nhất 1 phương thức và không phải các phương thức từ Object ( kể cả đã Override).

1. Sử dụng Functional Interface:

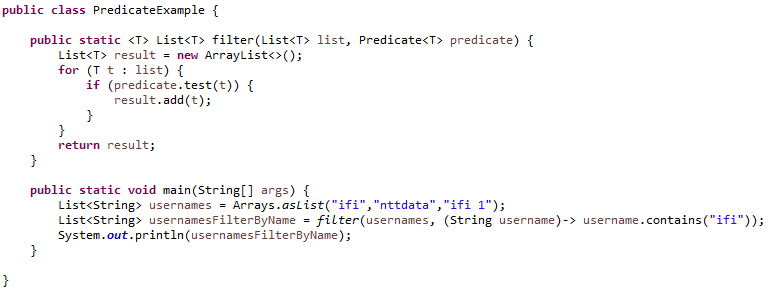
Một vài các Functional Interface của Java 8 cung cấp trong java.util.function



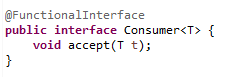
* Predicate:



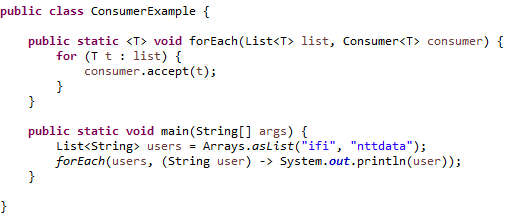
Implement:



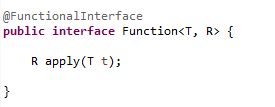
* Consumer:



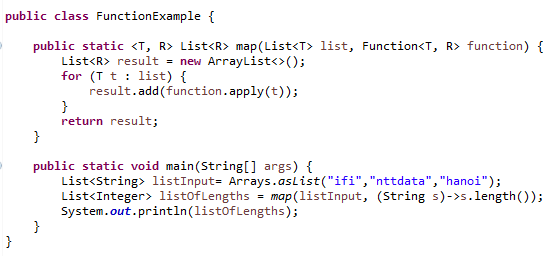
Implement:



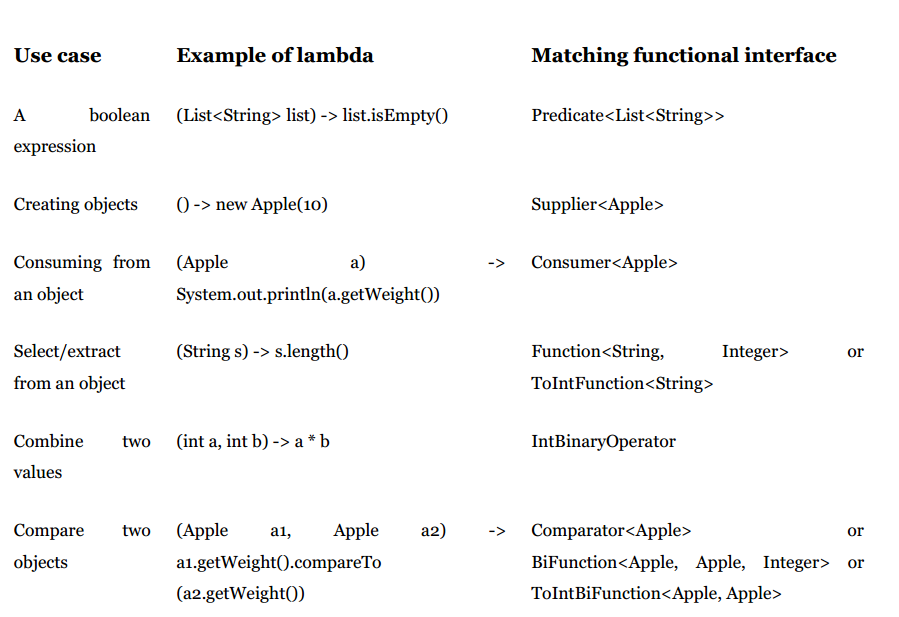
* Function:



Implement:



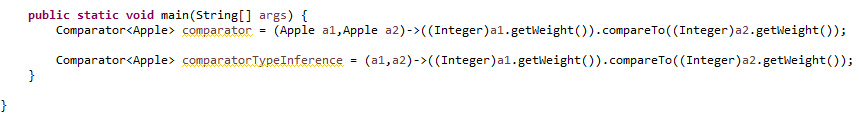
Các dạng cơ bản của Functional Interface.



* Kết luận:

**Lambda expression là một Functional interface implement.**

* **Type inference:**



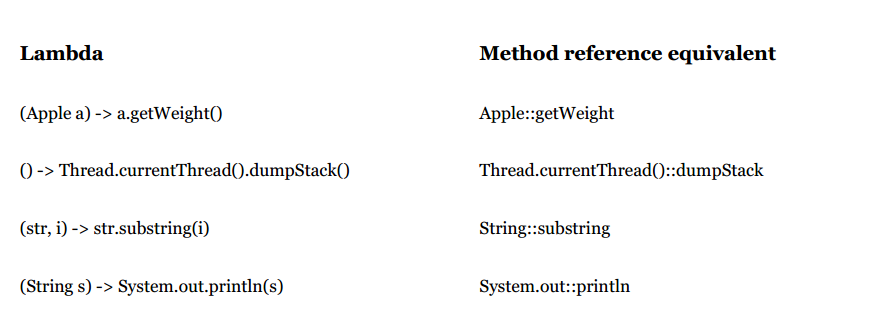
**Hai kiểu trên là giống nhau.**

**Java 8 tự suy luận ra kiểu của các tham số truyền vào.**

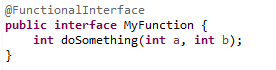
* Lambda là Closure, các biến bên trong nằm cùng 1 scope.

Nó có thể nhận biến ngoài tham gia vào body. Nhưng các biến ngoài phải là final.

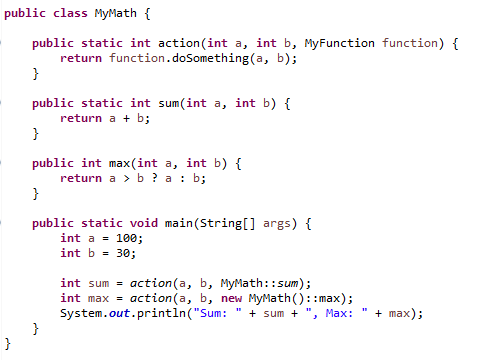
1. Method reference:

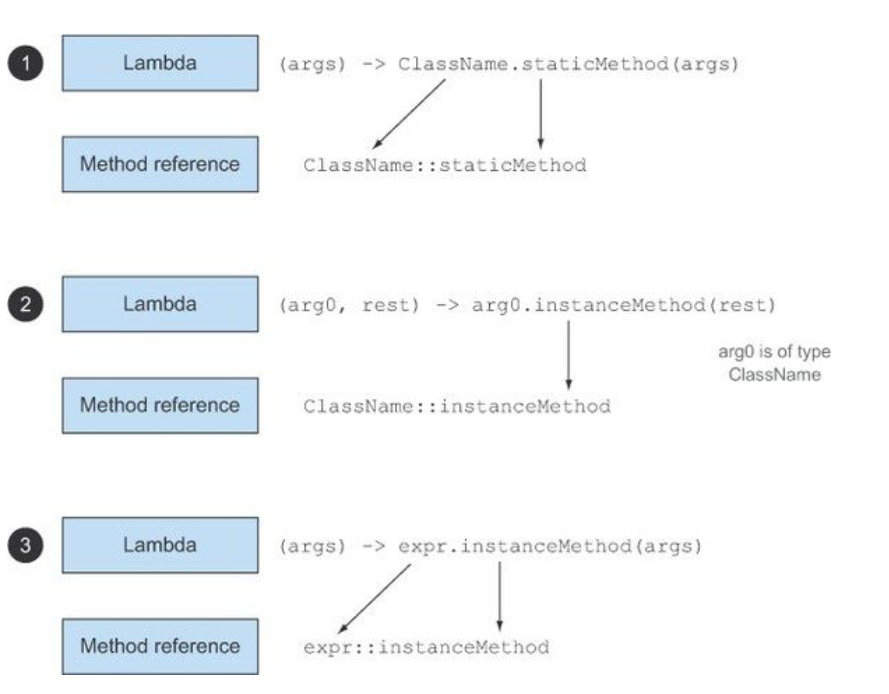


Ex:



Implement:





1. Constructor reference:

