

**Long.compare(long x, long y)**: So sánh hai giá trị kiểu long. Trả về 0 nếu chúng bằng nhau, một giá trị âm nếu x nhỏ hơn y, và một giá trị dương nếu x lớn hơn y.

**Long.compareUnsigned(long x, long y)**: So sánh hai giá trị kiểu long theo cách không dấu (unsigned). Thích hợp khi làm việc với số nguyên không dấu.

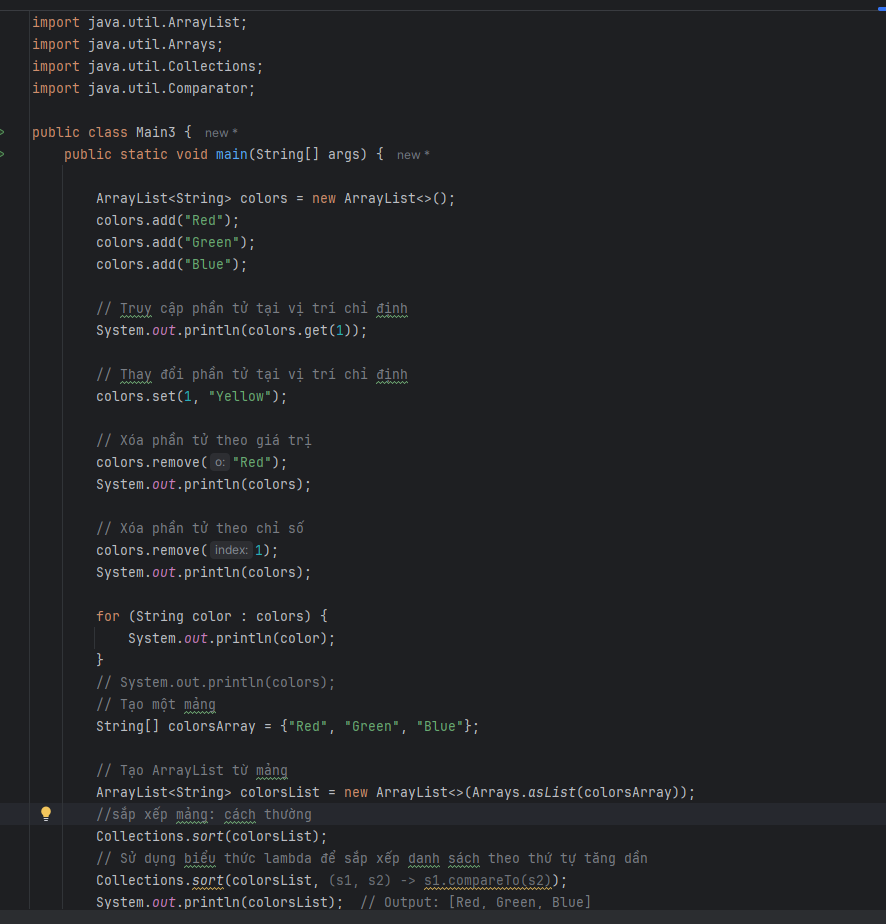
Collection:

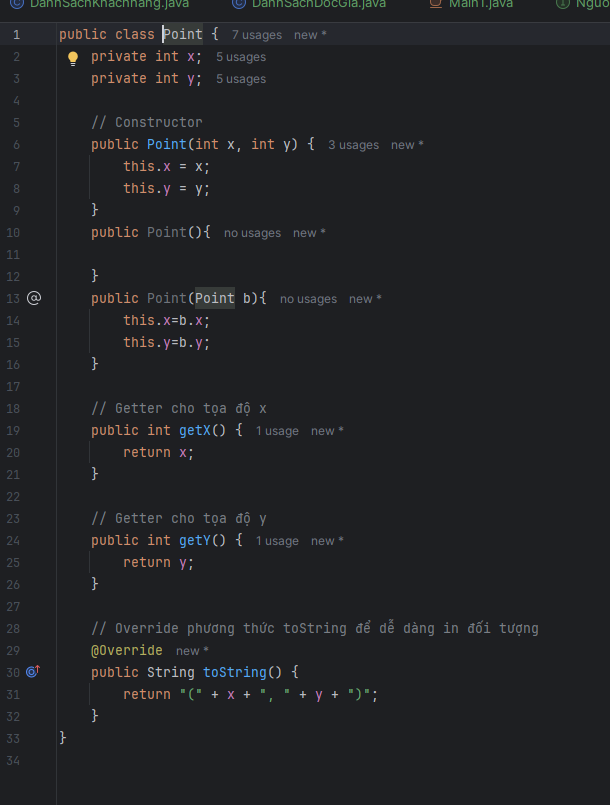
* Trong Java, có một số loại collection (tập hợp) cơ bản mà bạn có thể sử dụng để lưu trữ và quản lý dữ liệu

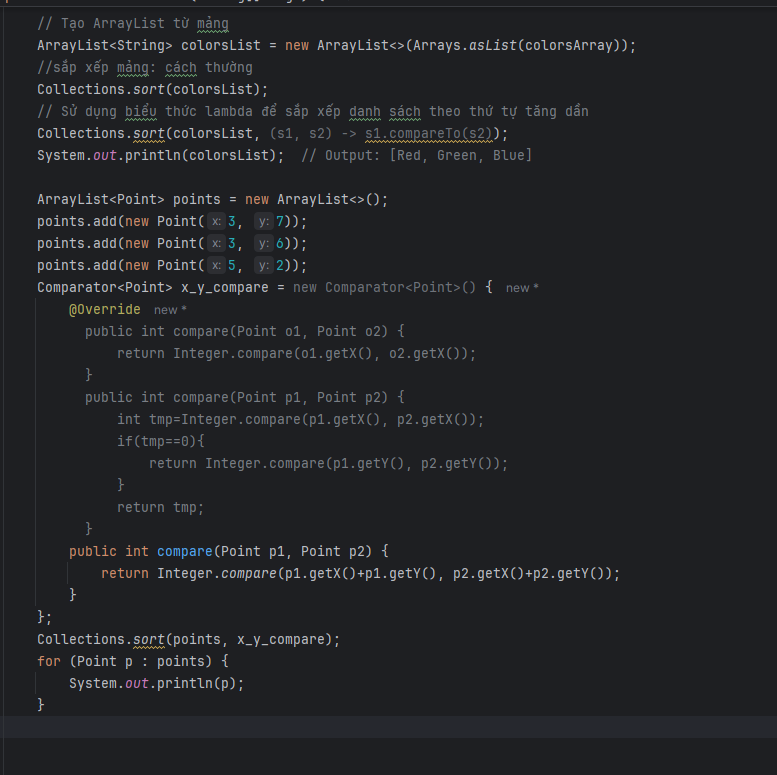
1.**List**

* List là một tập hợp có thứ tự và cho phép các phần tử trùng lặp. Các triển khai phổ biến của List bao gồm:
* **ArrayList**: Cung cấp hiệu suất tốt khi truy cập ngẫu nhiên và có khả năng mở rộng động.
* Các ví dụ:

Các ví dụ:







1. Định nghĩa: Biểu thức lambda trong Java (còn được gọi là lambda expression) là một tính năng được giới thiệu từ Java 8, giúp bạn viết mã ngắn gọn hơn và dễ đọc hơn khi làm việc với các đối tượng như Runnable, Comparator, Function, Predicate, và nhiều loại functional interface khác.
2. Ưu điểm: Biểu thức lambda mang lại nhiều lợi ích như giảm độ dài mã, cải thiện tính đọc hiểu, hỗ trợ lập trình hàm, tăng tính tái sử dụng và hiệu suất. Sử dụng lambda expressions là một cách hiệu quả để viết mã sạch hơn và dễ bảo trì hơn trong các ứng dụng Java hiện đại.
3. Nội dung: các biểu thức lambda thường được sử dụng với các functional interfaces để đơn giản hóa mã và cải thiện khả năng đọc. Các functional interfaces phổ biến bao gồm Predicate, Function, Consumer, và Supplier.
4. Predicate:

+Sử dụng Predicate khi bạn cần thực hiện một kiểm tra hoặc điều kiện và trả về giá trị boolean.

+**Phương Thức Chính**: boolean test(T t);

**+ VD Cú Pháp Lambda**: Predicate<String> isNotEmpty = s -> !s.isEmpty();

1. Function:

+Là một functional interface nhận một đối tượng đầu vào và trả về một kết quả. Có thể dùng để biến đổi hoặc xử lý dữ liệu.

+**Phương Thức Chính**: R apply(T t);

**+VD Cú Pháp Lambda**: Function<String, Integer> stringLength = s -> s.length();

+Sử dụng Function khi bạn cần chuyển đổi hoặc xử lý dữ liệu và nhận được kết quả là một đối tượng khác.

1. Consumer

+ Consumer là một functional interface nhận một đối tượng đầu vào và thực hiện một hành động trên đối tượng đó mà không trả về giá trị.

+**Phương Thức Chính**: void accept(T t);

**+VD Cú Pháp Lambda**: Consumer<String> printUpperCase = s -> System.out.println(s.toUpperCase());

+Sử dụng Consumer khi bạn cần thực hiện một hành động trên dữ liệu mà không cần trả về giá trị.

1. Stream

* Stream không phải là một functional interface mà là một API trong Java 8 để làm việc với tập hợp dữ liệu theo cách hàm (functional style). Streams cho phép bạn thực hiện các phép toán trên dữ liệu, chẳng hạn như lọc, ánh xạ, và sắp xếp.
* **Phương Thức Chính**: Stream không có một phương thức chính duy nhất; thay vào đó, nó cung cấp nhiều phương thức như filter, map, reduce, v.v.
* **Cú Pháp Lambda**: Sử dụng các biểu thức lambda trong các phép toán của stream.
* Sử dụng Stream khi bạn muốn xử lý dữ liệu tập hợp theo cách hàm, cho phép thực hiện các phép toán phức tạp trên dữ liệu một cách dễ dàng và hiệu quả.
* Các hàm:  
  ***Tạo Stream****: Bạn có thể tạo một stream từ các tập hợp dữ liệu như danh sách, mảng, hoặc tạo stream từ các giá trị cụ thể.*

***Lọc****: Sử dụng filter để giữ lại các phần tử thỏa mãn điều kiện.*

***Ánh Xạ****: Sử dụng map để chuyển đổi từng phần tử trong stream.*

***Sắp Xếp****: Sử dụng sorted để sắp xếp các phần tử trong stream.*

***Tính Toán****: Sử dụng các phương thức như sum, average, max, min để thực hiện các phép toán tổng hợp.*

***Kết Hợp Phép Toán****: Sử dụng nhiều phép toán kết hợp để xử lý dữ liệu theo cách mong muốn.*

***FlatMap****: Kết hợp các danh sách con thành một danh sách duy nhất.*

***Giảm****: Sử dụng reduce để thực hiện các phép toán tổng hợp hoặc tính toán phức tạp.*

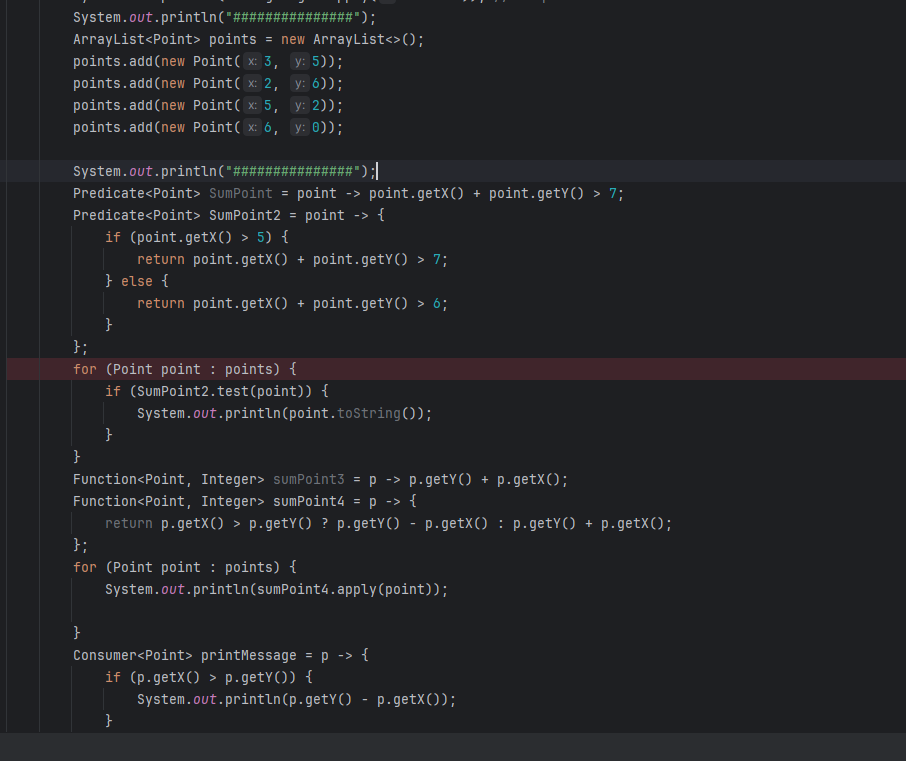
VD: Biểu thức lambda để tính bình phương của một số Function<Integer, Integer> square = x -> x \* x; // Sử dụng biểu thức lambda

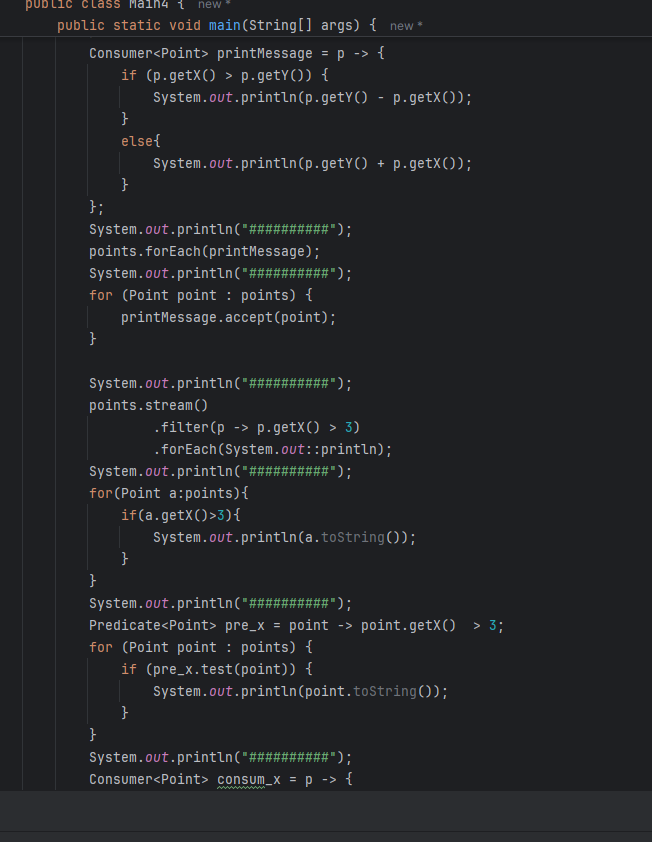
System.out.println(square.apply(5)); // Output: 25

Ví dụ 2:

// Biểu thức lambda để kiểm tra xem một số có phải là số chẵn không Predicate<Integer> isEven = num -> { return num % 2 == 0; };

System.out.println(isEven.test(4)); // Output: true System.out.println(isEven.test(5)); // Output: false

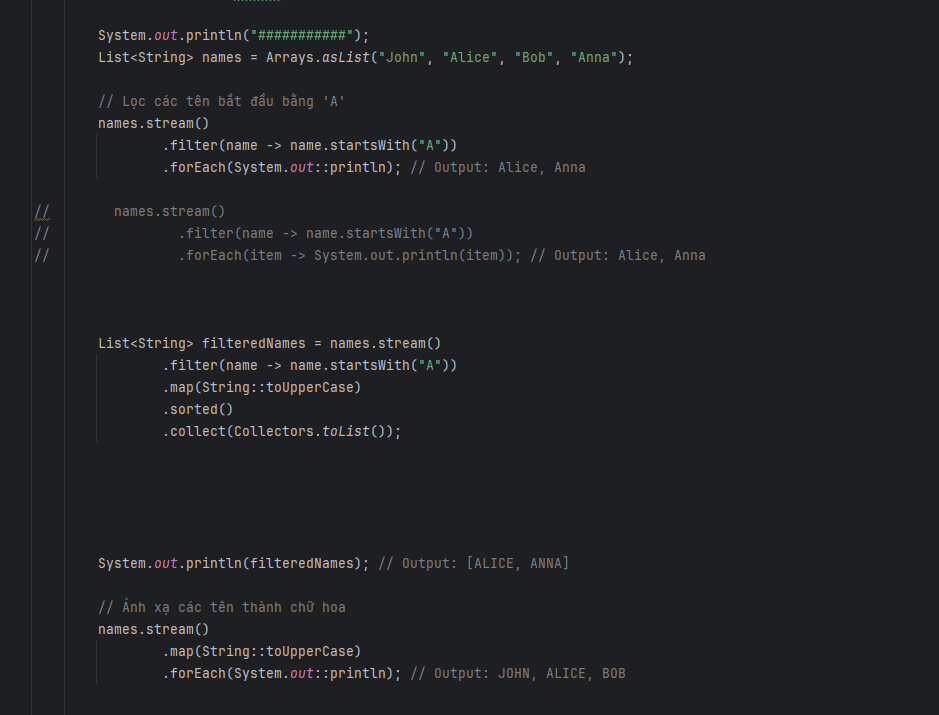


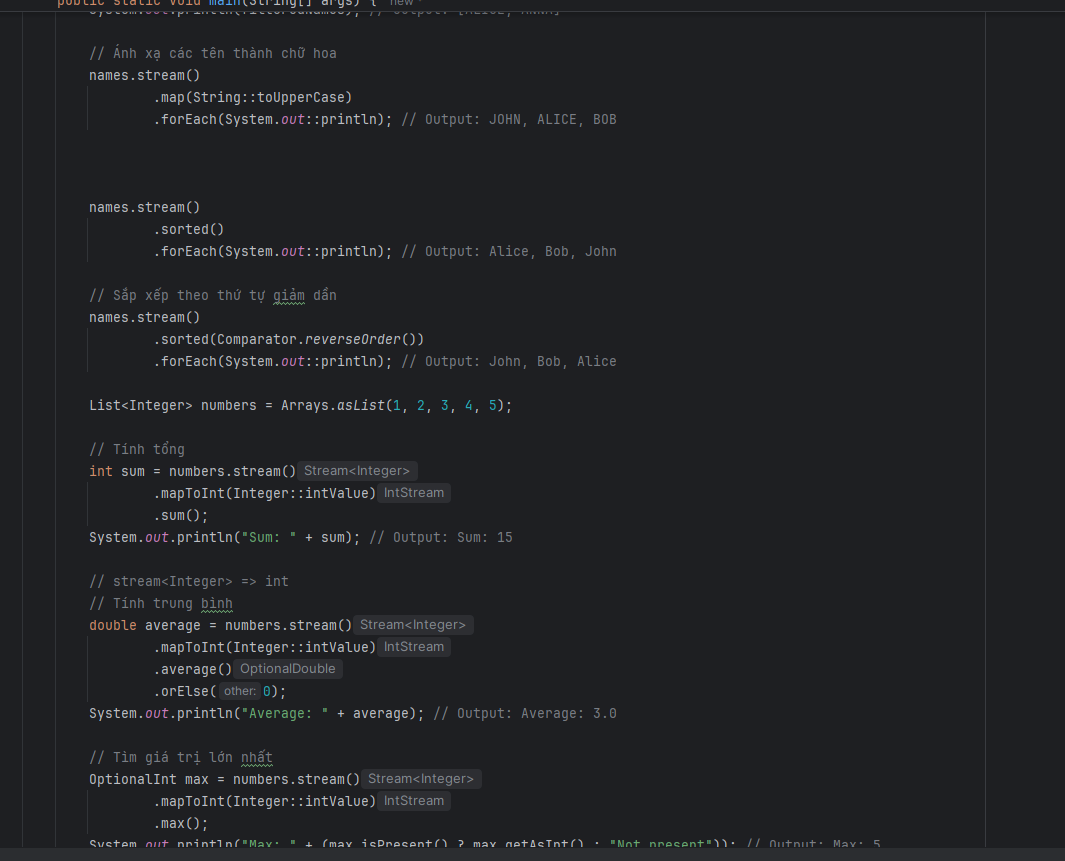


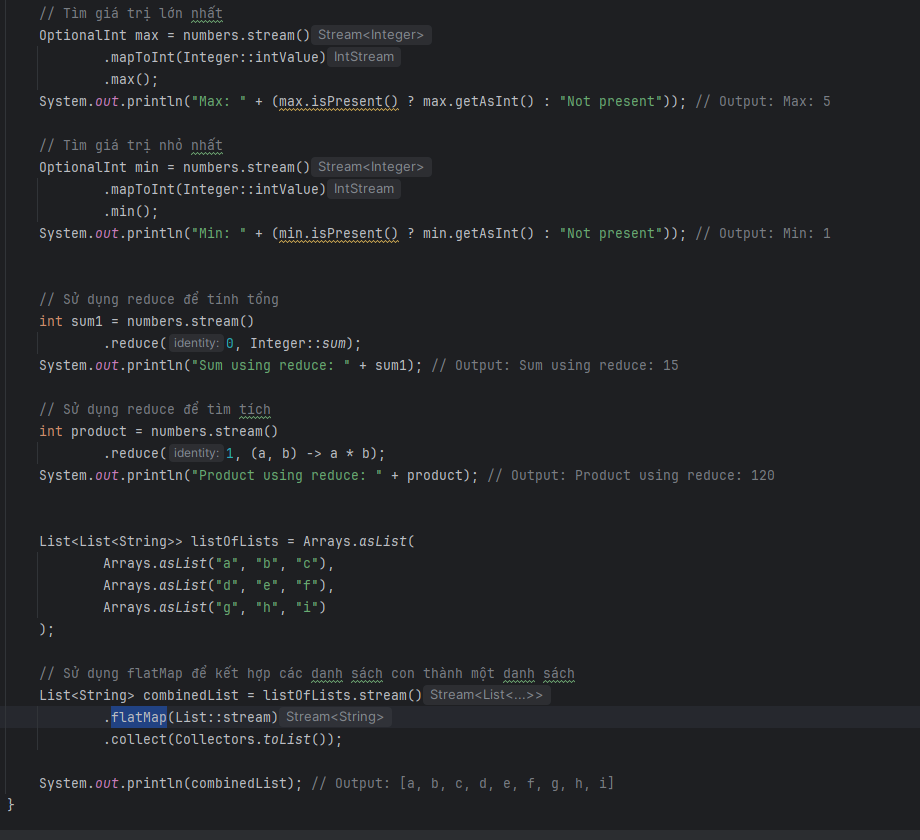


Việc chuyển đổi từ Stream<Integer> thành IntStream giúp bạn:

* **Tăng hiệu suất**: Vì làm việc với kiểu dữ liệu nguyên thủy (int) hiệu quả hơn so với đối tượng bao bọc (Integer).
* **Sử dụng các phương thức số học tích hợp sẵn**: Như sum(), average(), và các phép toán khác mà IntStream cung cấp.
* **Giảm tiêu tốn bộ nhớ**: Bằng cách tránh việc lưu trữ các đối tượng bao bọc không cần thiết.







Ví dụ kiểu point

* Viết 4 từ tiếng anh: kế thừa, đa hình, trừa tượng, đóng gói

Lê Khoa,Hoàng Hảo,ngọc Hải,Ngọc Nguyên

* Tính tổng các x,y của danh sách:

+ngọc đạt,Văn Hoàng,Hoàng Minh

-3 bạn giải đúng: Khoa, Hoàng, Sỹ

