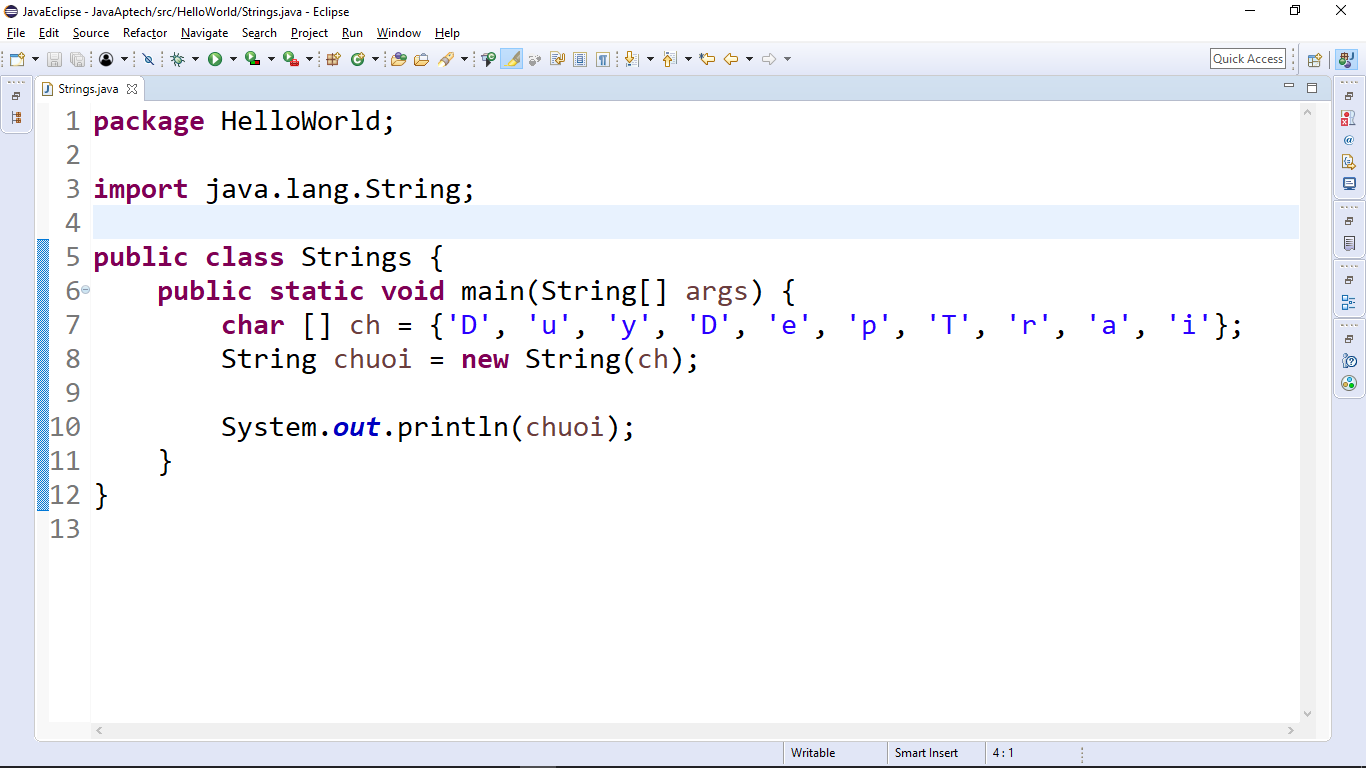
LỘ TRÌNH ĐÀO TẠO PHẦN 5, 6, 7

1. Làm quen với chuỗi
   1. **Làm việc với String**
      1. Chuỗi là gì ?

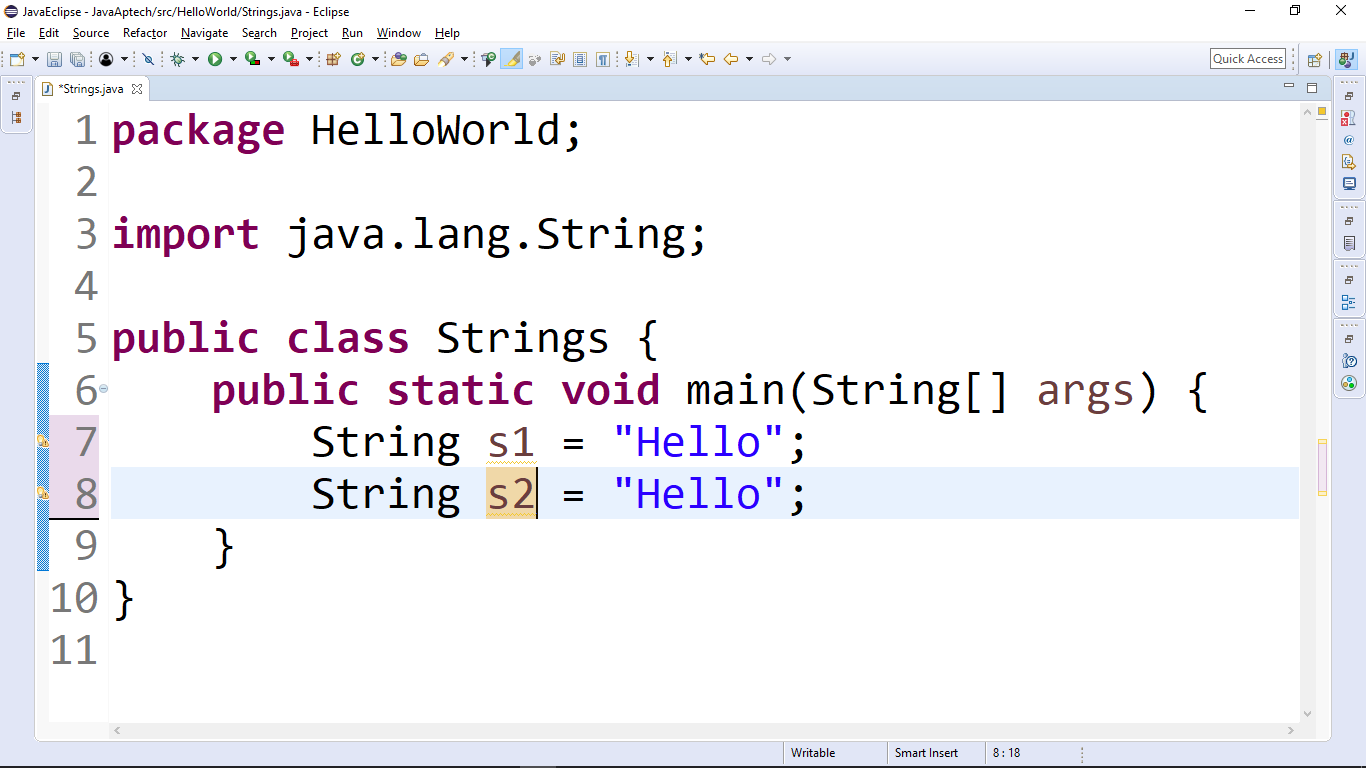
* Chuỗi là một dãy ký tự liên tục. Trong Java, String là một đối tượng mà biểu diễn một dãy ký tự liên tục.
* Cơ bản về chuỗi là một đối tượng biển diễn dãy các kí tự.

VD:



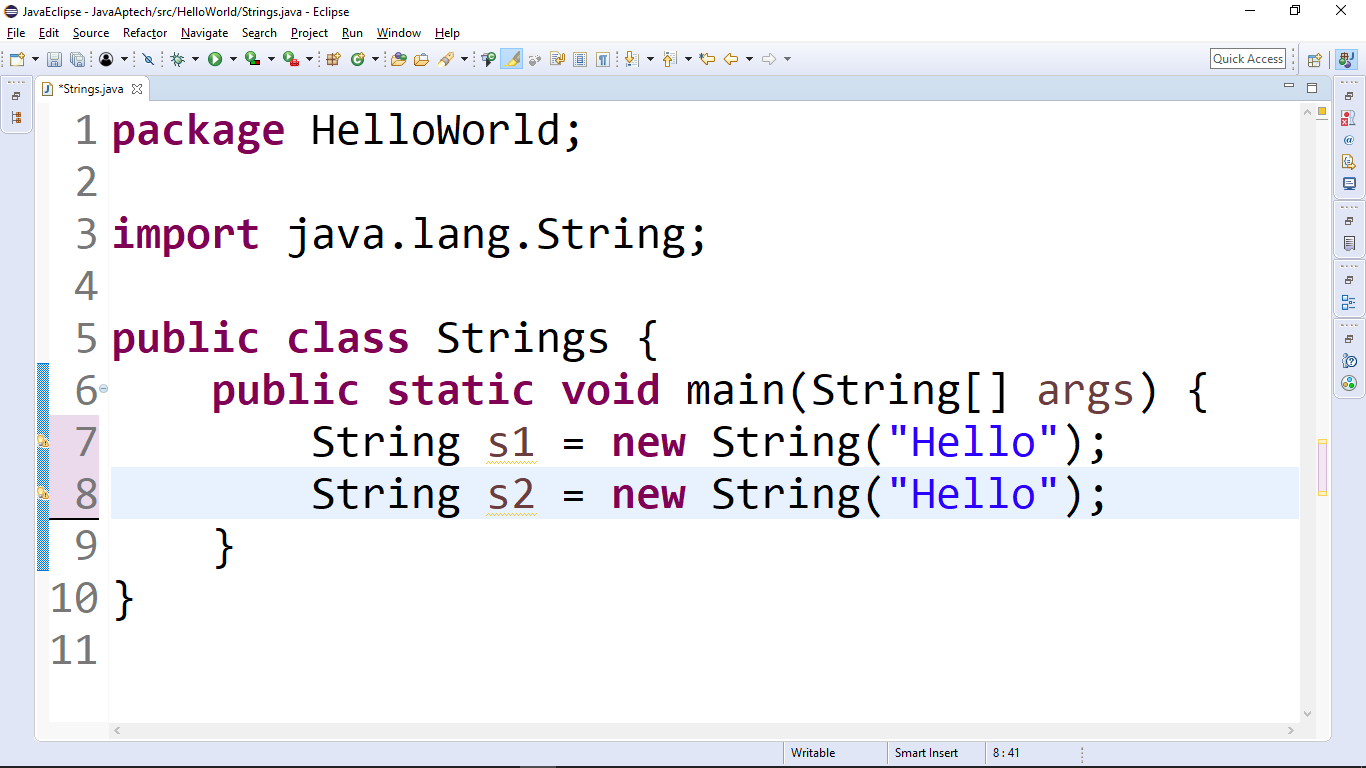
* + 1. Tạo đối tượng với lớp String.
       1. Tạo không có tử khóa **new**
* Khi bạn tạo một hằng chuỗi, đầu tiên JVM kiểm tra đã có có hằng chuỗi đó chưa. Nếu chuỗi đã tồn tại, nó sẽ tham chiếu đến và trả về giá trị đã có. Nếu chuỗi không tồn tại, một chuỗi mới được tạo ra.

VD:

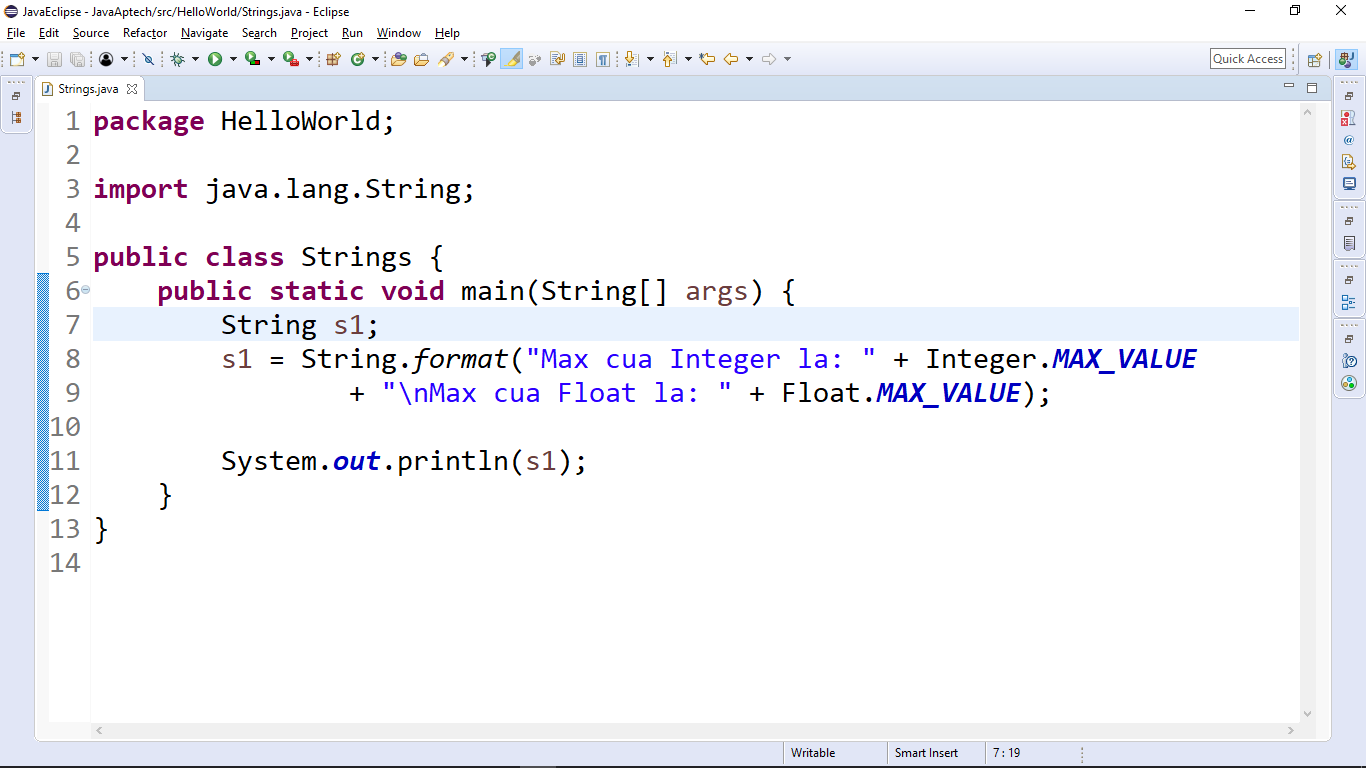


* Trong ví dụ trên chỉ có một đối tượng sẽ được tạo. Đầu tiên JVM sẽ không tìm thấy bất cứ đối tượng String nào với giá trị "xinchao", vì thế nó sẽ tạo một đối tượng mới. Sau đó, nó sẽ tìm thấy chuỗi với giá trị "xinchao", nó sẽ không tạo đối tượng mới nhưng sẽ trả về tham chiếu tới cùng instance (sự thể hiện) của đối tượng đó.
  + - 1. Tạo có từ khóa **new**
* Khi sử dụng từ khóa **new** JVM sẽ tạo ra một đối tượng mới, bao nhiêu **new** sẽ có bao nhiêu đối tượng\

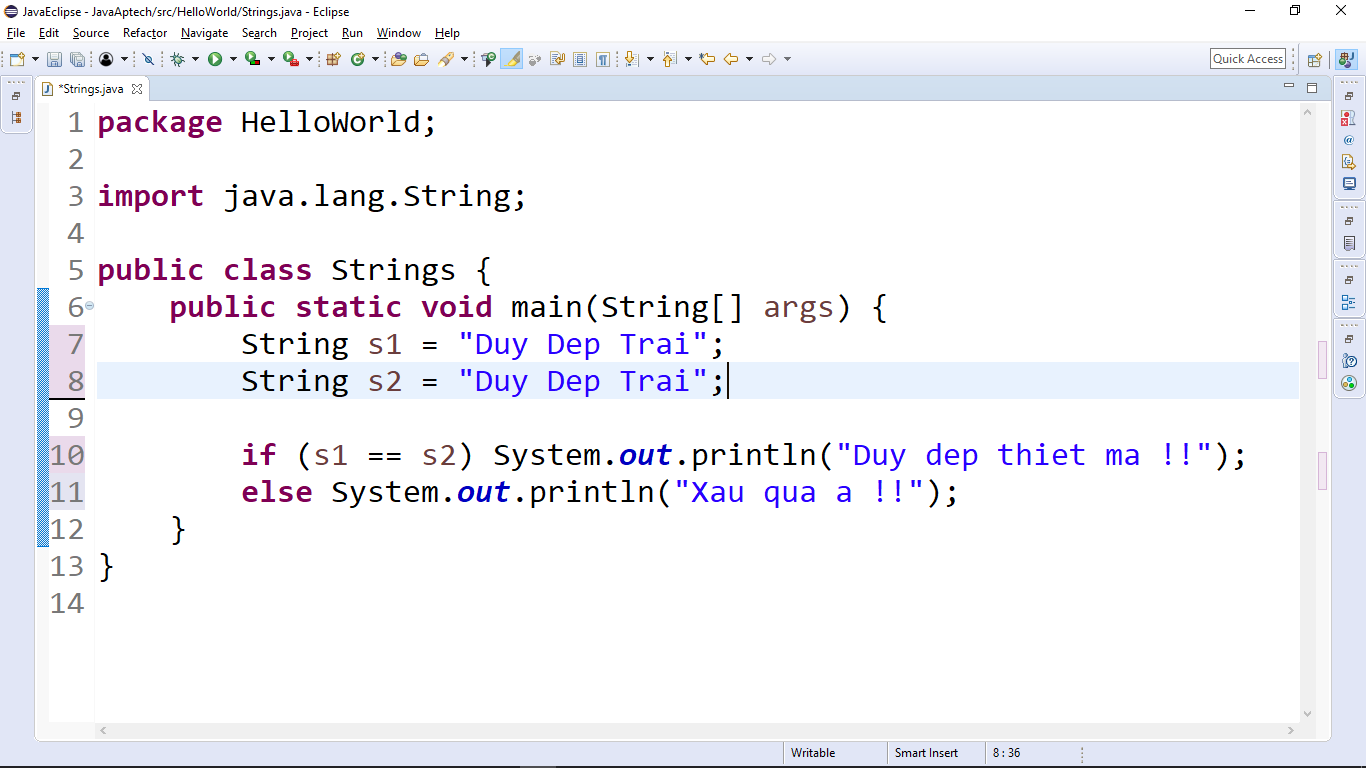
VD:

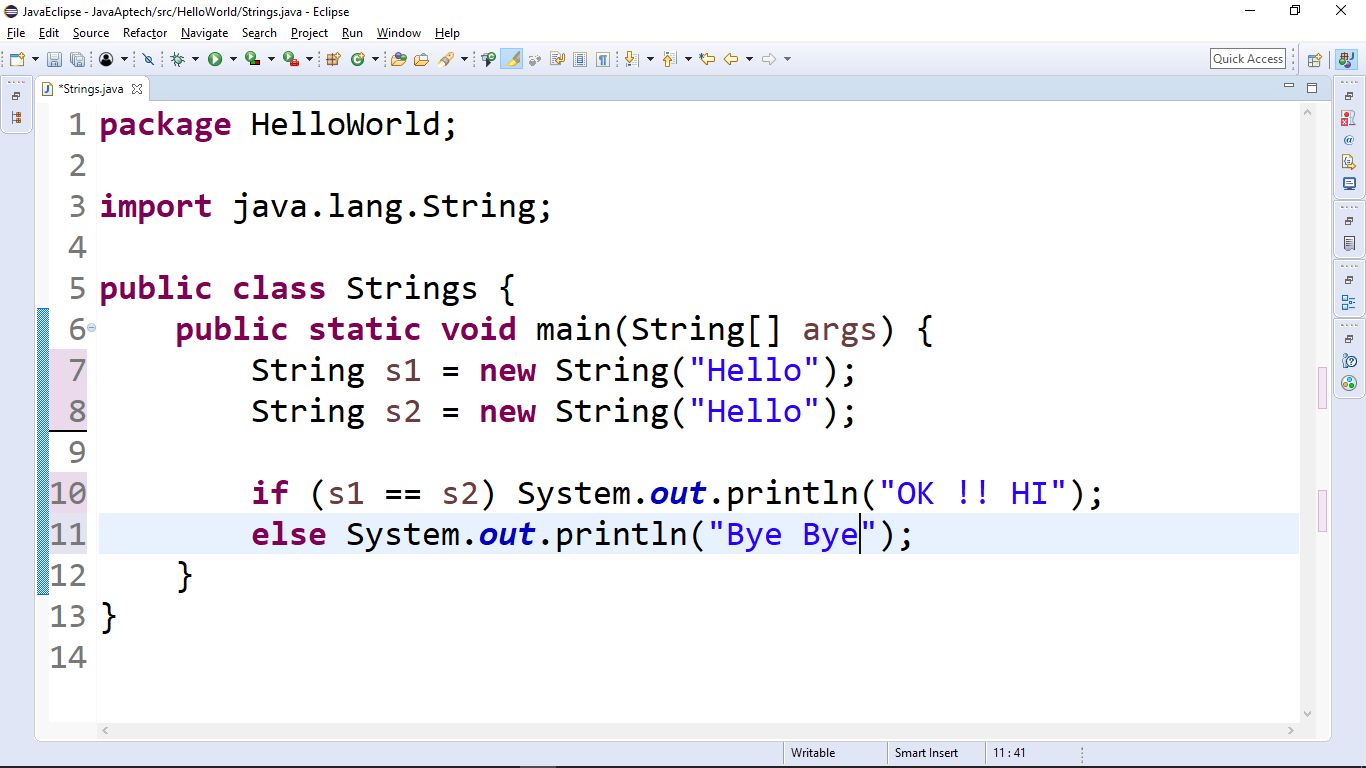


**NOTE**: Khi sử dùng phương thức println() để output ra ngoài thì ta tất cả các thông tin trong đó đều được định dạng thành String. Trong String cũng có phương thức tương đương là format().

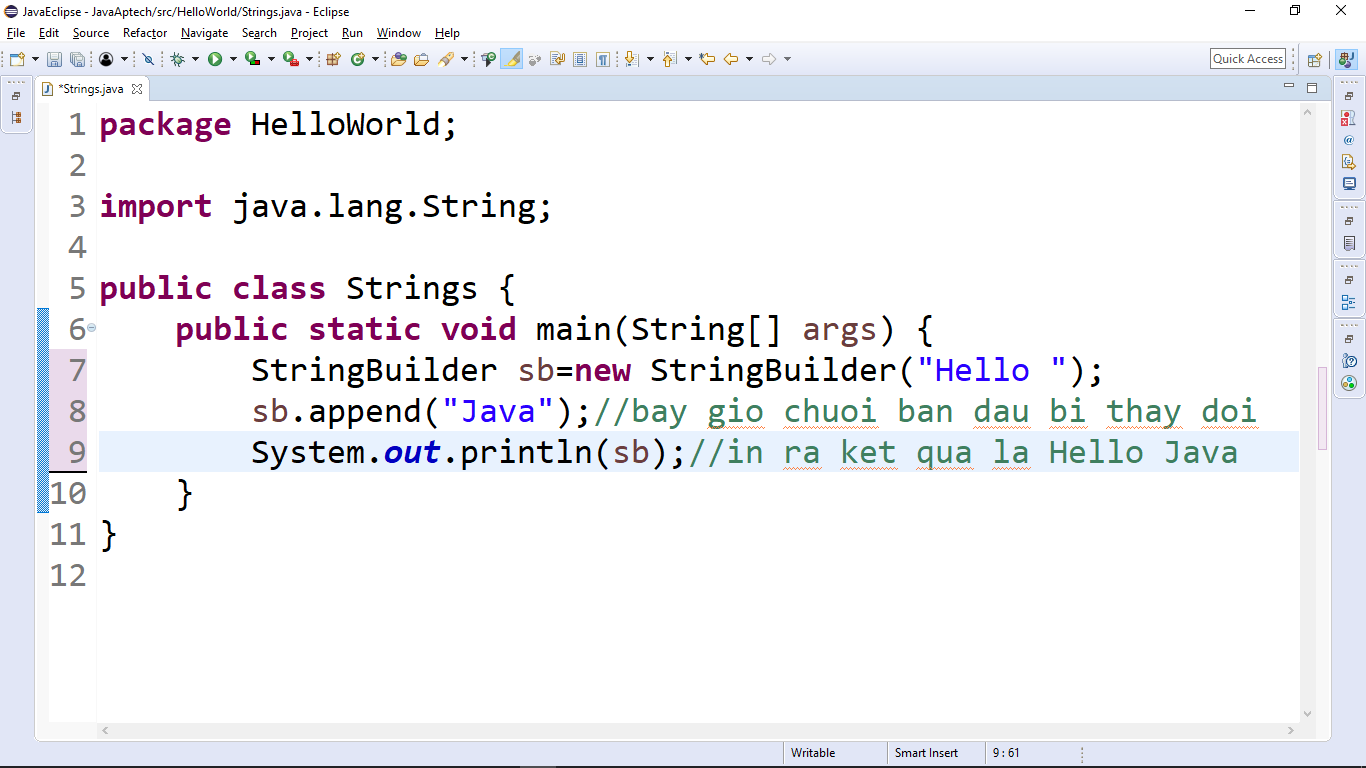


* + 1. So sánh chuỗi.
* Cho 2 ví dụ đưa ra kết luận.

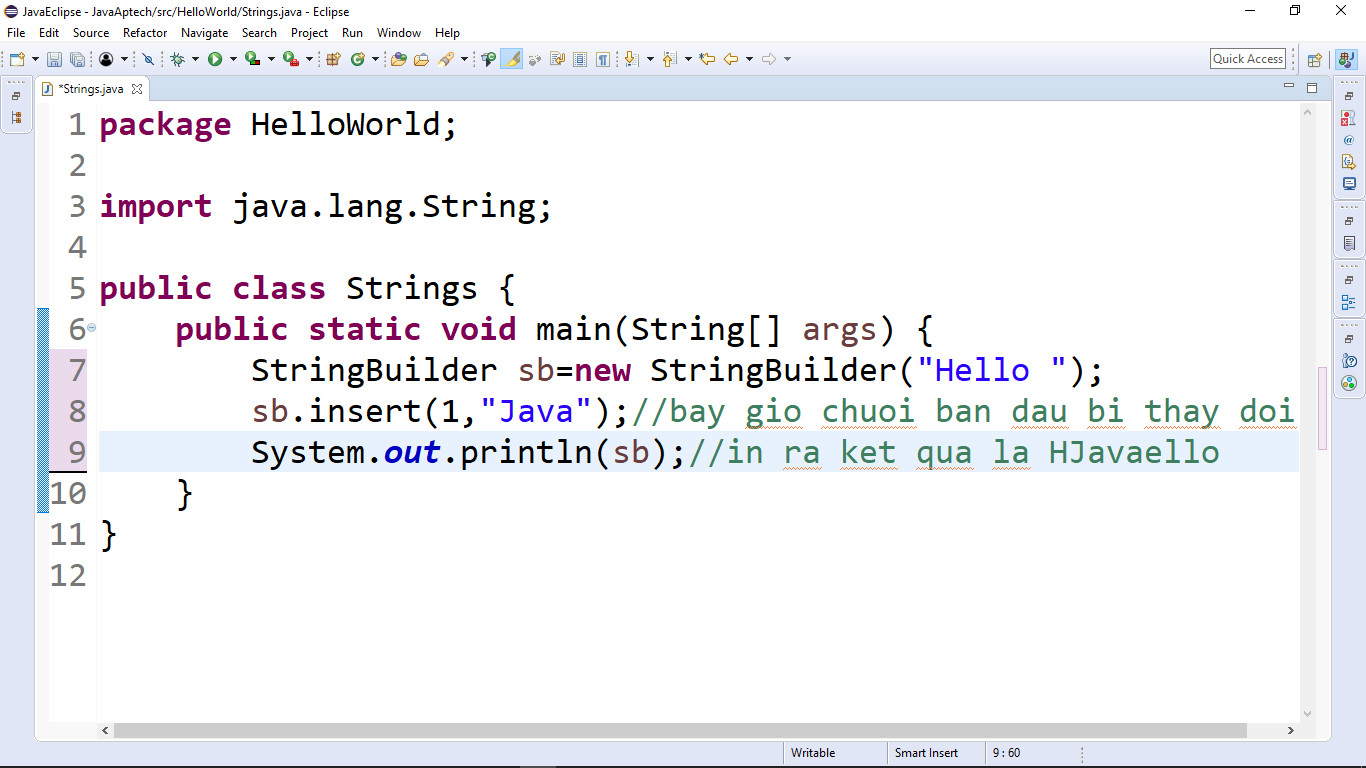




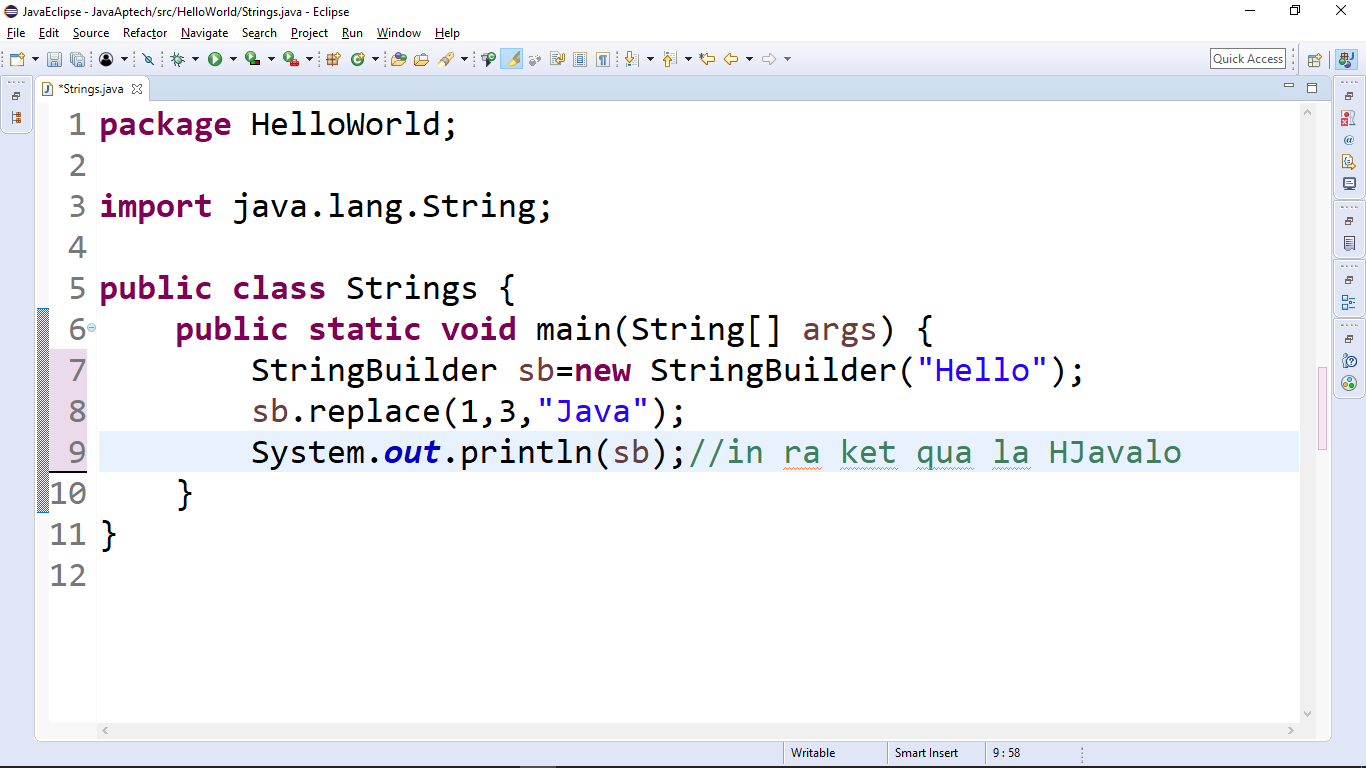
* 1. **Giới thiệu StringBuilder**
     1. Định nghĩa
* StringBuilder là không đồng bộ (*non-synchronized*) tức là không an toàn luồng. Nghĩa là hai Thread có thể gọi đồng thời các phương thức của lớp StringBuilder. Hay còn gọi là đa luồng.
  + 1. Một vài phương thức hay dùng
       1. **append()**
* Dùng để nối chuỗi.



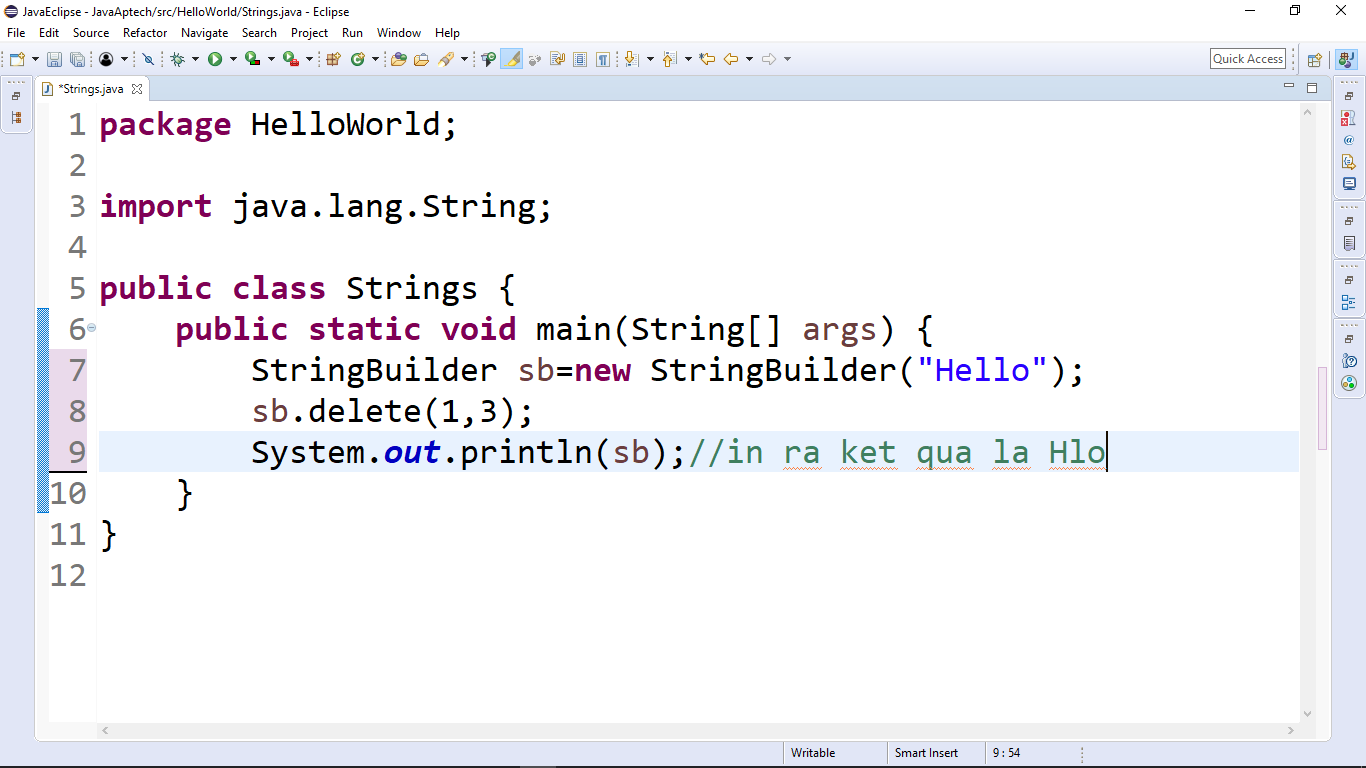
* + - 1. **insert()**
* Phương thức insert() chèn chuỗi đã cho vào chuỗi này tại vị trí đã cho.



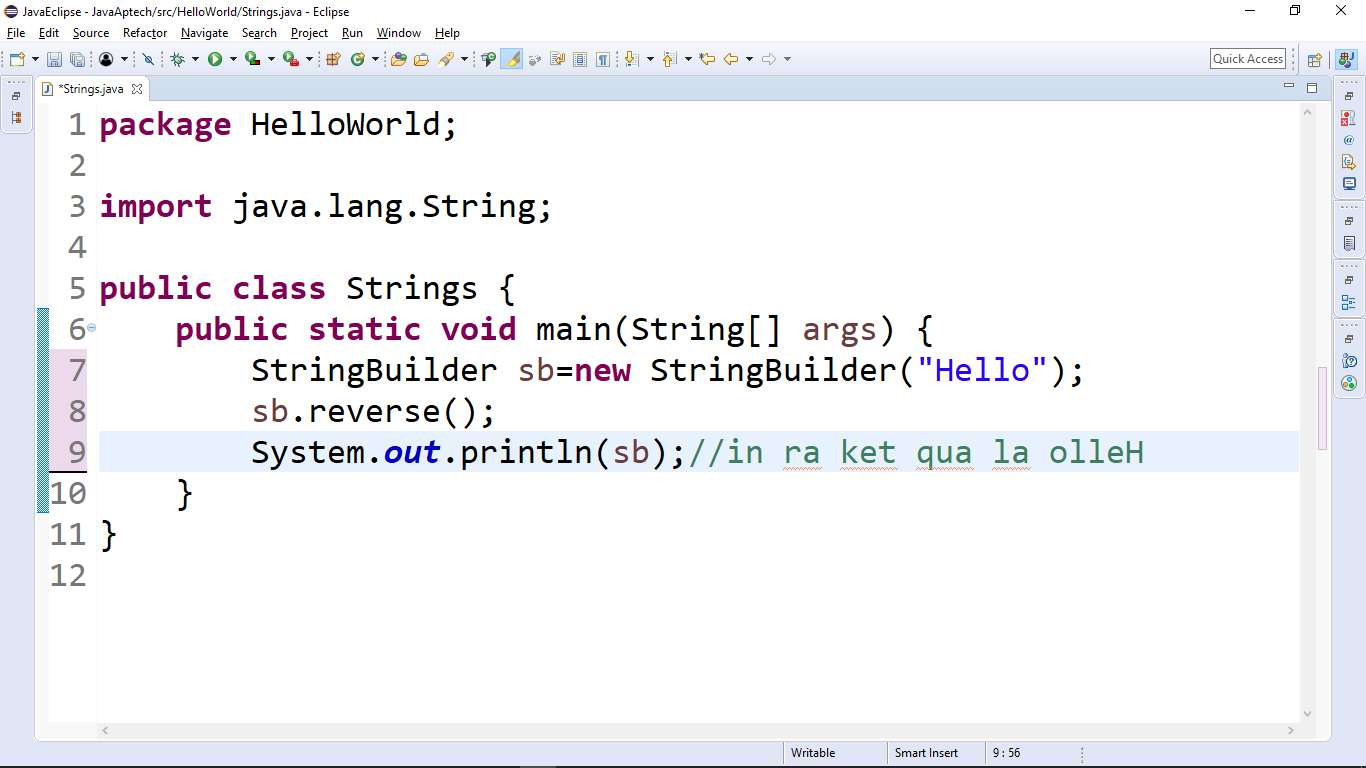
* + - 1. **replace()**
* Phương thức replace() thay thế chuỗi đã cho bắt đầu từ chỉ mục beginIndex tới endIndex



* + - 1. **delete()**
* Phương thức delete() xóa chuỗi bắt đầu từ chỉ mục beginIndex tới endIndex đã cho.



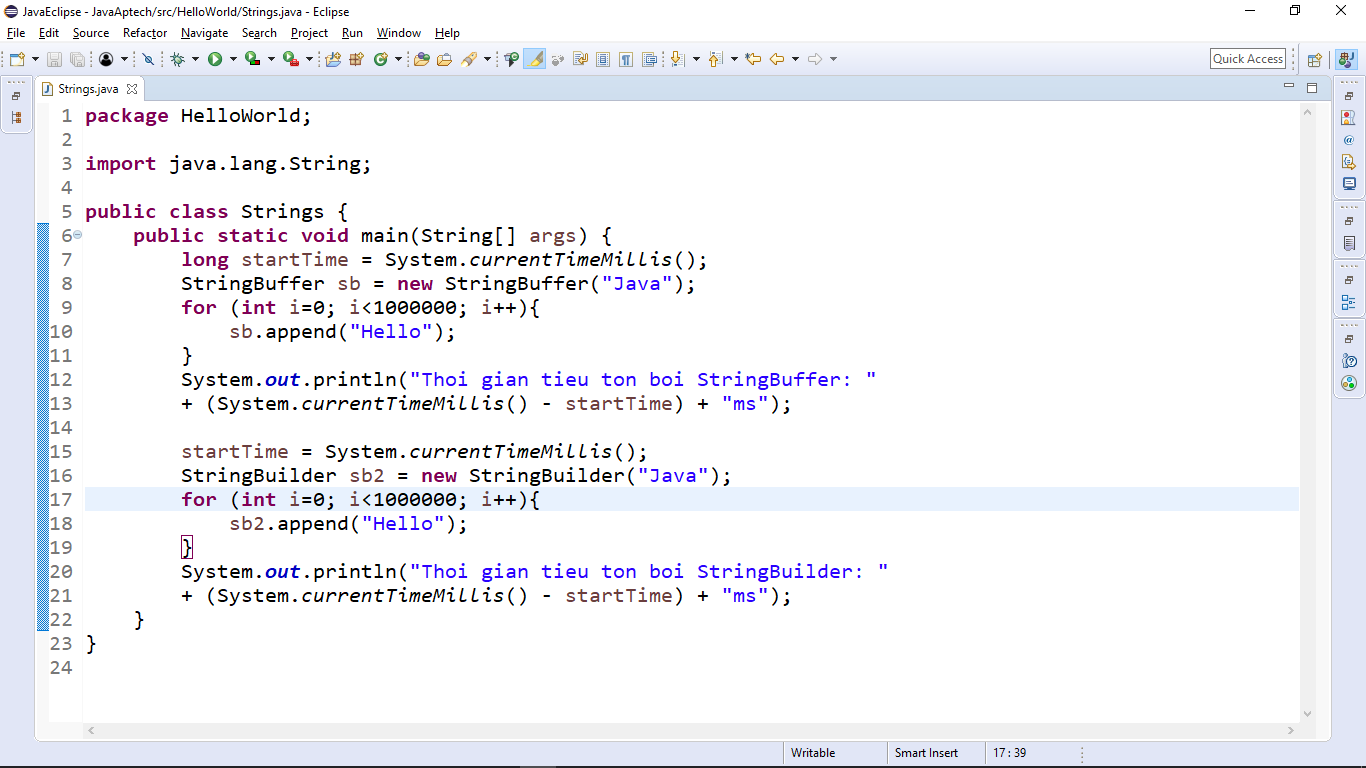
* + - 1. **reverse()**
* Phương thức reverse() đảo ngược chuỗi hiện tại.



* 1. **Giới thiệu StringBuffer**
     1. Định nghĩa
* Lớp StringBuffer là đồng bộ (*synchronized*), tức là an toàn luồng (thread safe). Nghĩa là hai Thread không thể gọi đồng thời các phương thức của lớp StringBuffer
  + 1. Một vài phương thức hay dùng (Giống StringBuilder)
  1. **So sánh StringBuilder và StringBuffer**

|  |  |
| --- | --- |
| **Lớp StringBuffer** | **Lớp StringBuilder** |
| Lớp StringBuffer là đồng bộ (*synchronized*), tức là an toàn luồng (thread safe). Nghĩa là hai Thread không thể gọi đồng thời các phương thức của lớp StringBuffer | StringBuilder là không đồng bộ (*non-synchronized*) tức là không an toàn luồng. Nghĩa là hai Thread có thể gọi đồng thời các phương thức của lớp StringBuilder |
| StringBuffer là *kém hiệu quả hơn* StringBuilder | StringBuilder là *hiệu quả hơn* StringBuffer |

* 1. **Kiểm tra hiệu suất của StringBuffer và StringBuilder**

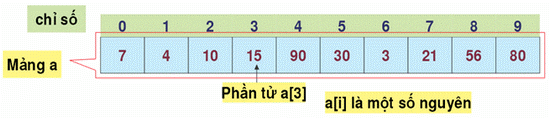


1. Làm quen với mảng
   1. Mảng 1 chiều
      1. **Đặt vấn đề.**

* Chương trình cần lưu trữ 3 số nguyên
* Khai báo 3 biến kiểu số nguyên
* Chương trình cần lưu trữ 100 số nguyên
* Khai báo 100 biến kiểu số nguyên
* Chương trình cần lưu trữ n số nguyên

? Phải làm thế nào để lưu trữ ???

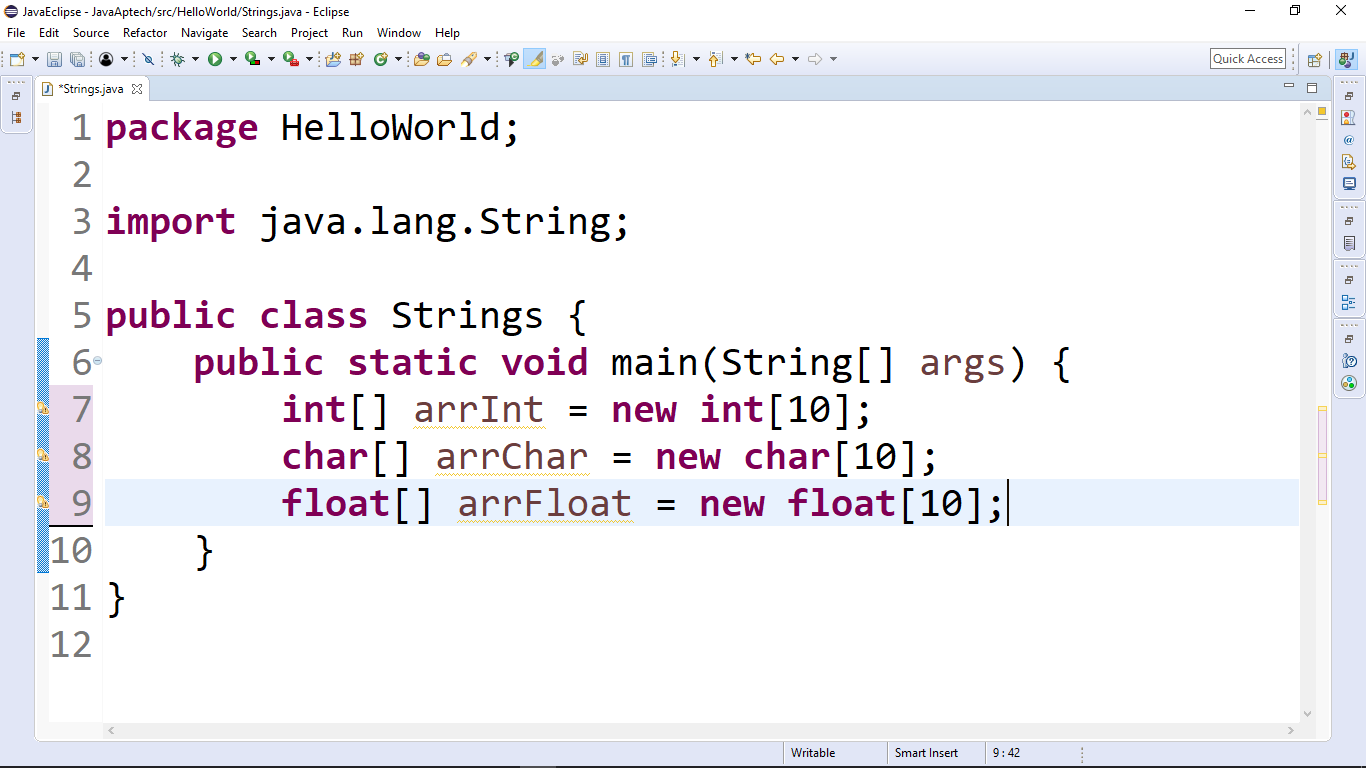
* + 1. **Khái niệm**
* Là kiểu dữ liệu có cấu trúc do người lập trình định nghĩa.
* Biểu diễn một dãy các biến cùng kiểu dữ liệu.
* Kích thước được xác định ngay lúc khai báo và không thay đổi.



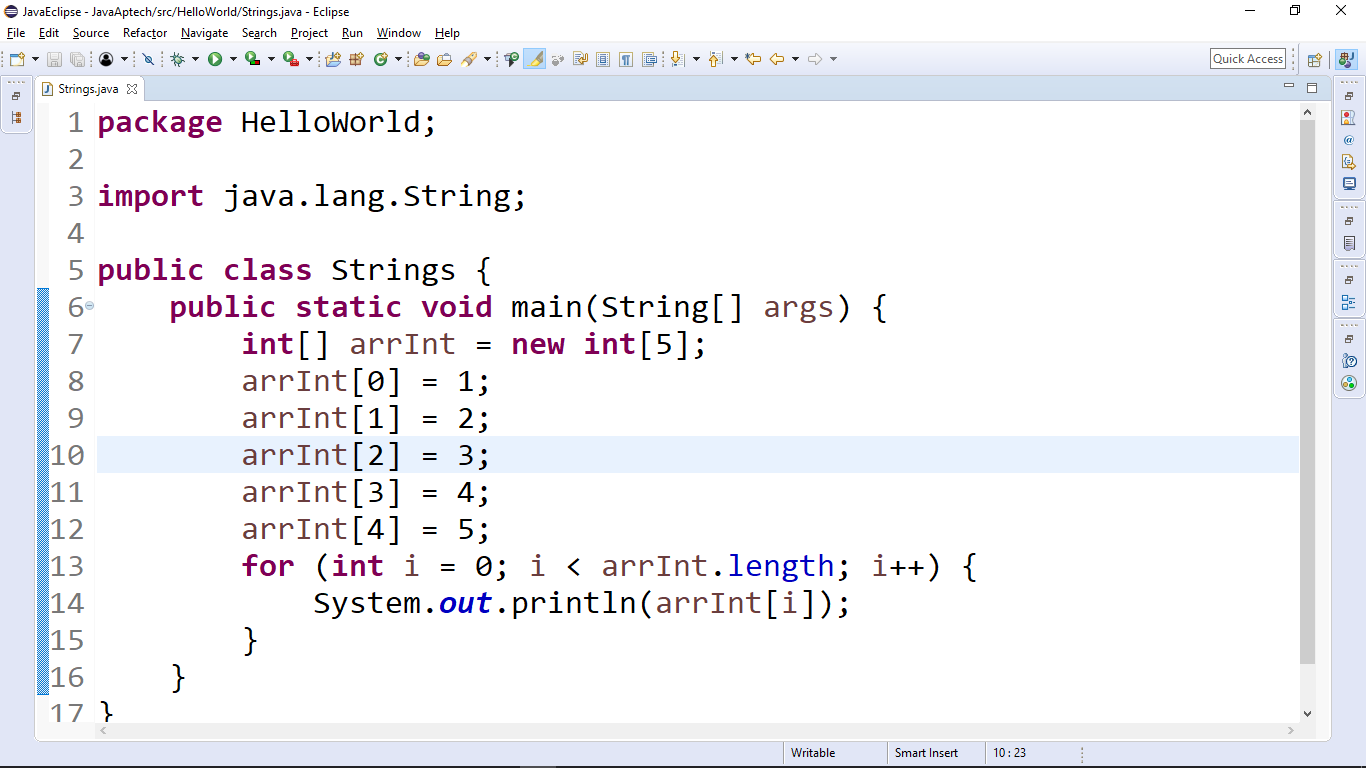
* + 1. **Khai báo và sử dụng mảng**
* <Kiểu dữ liệu> [] <tên biến mảng> = new <Kiểu dữ liệu> [<số phần tử>] hoặc

<Kiểu dữ liệu> <tên biến mảng>[] = new <Kiểu dữ liệu> [<số phần tử>].

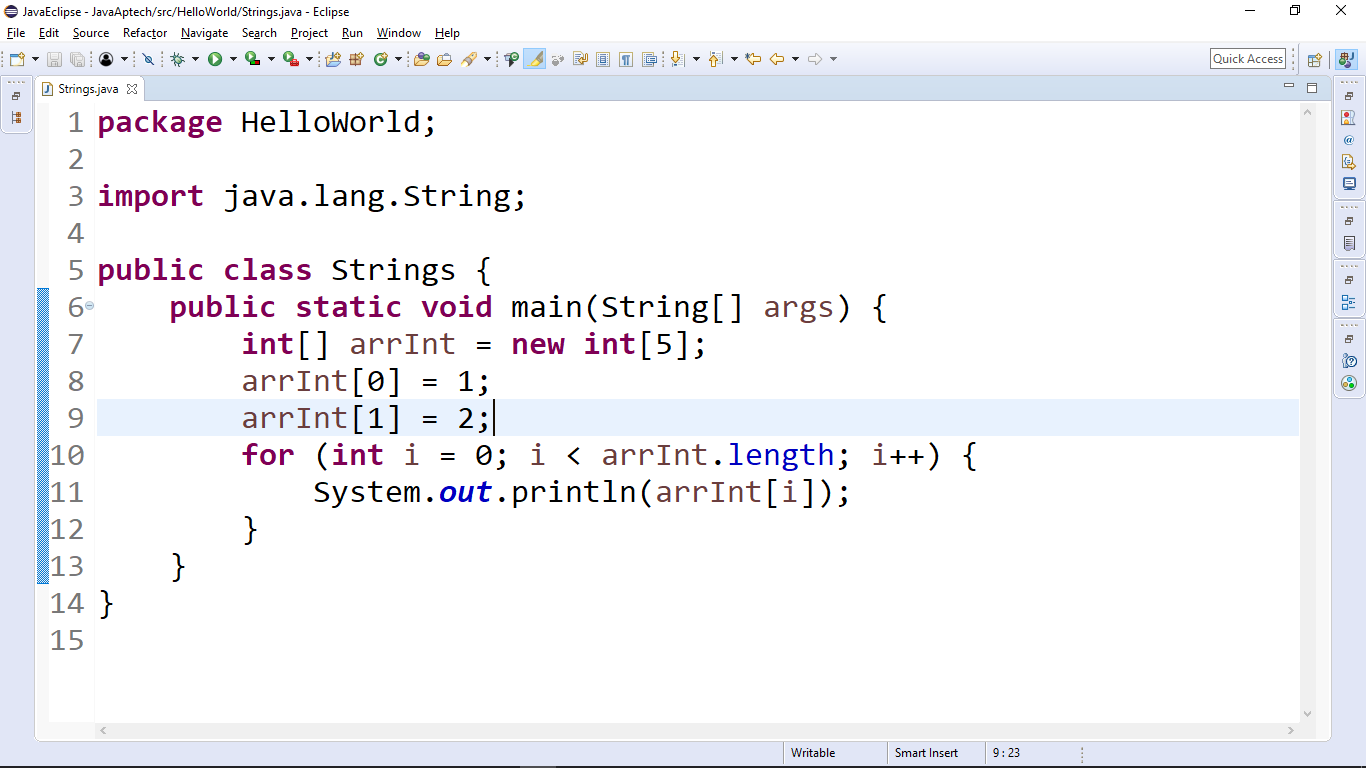
* Lưu ý:
  + Phải xác định <số phần tử> cụ thể (hằng) khi khai báo.
  + Mảng là 1 dãy liên tục có chỉ số bắt đầu bằng 0 đển **tổng số phần tử** - 1.



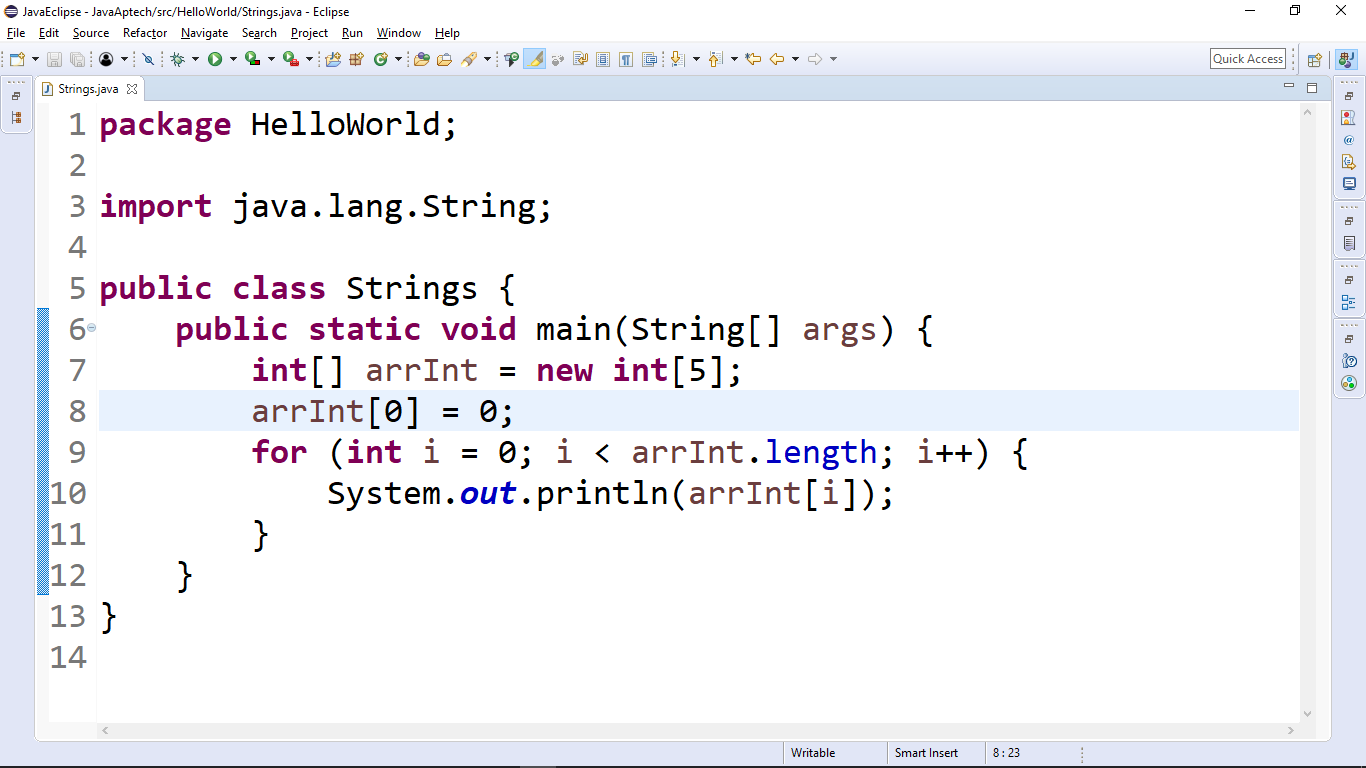
* + 1. Khai báo và gán giá trị ban đầu.
* Khởi tạo giá trị cho phần tử trong mảng



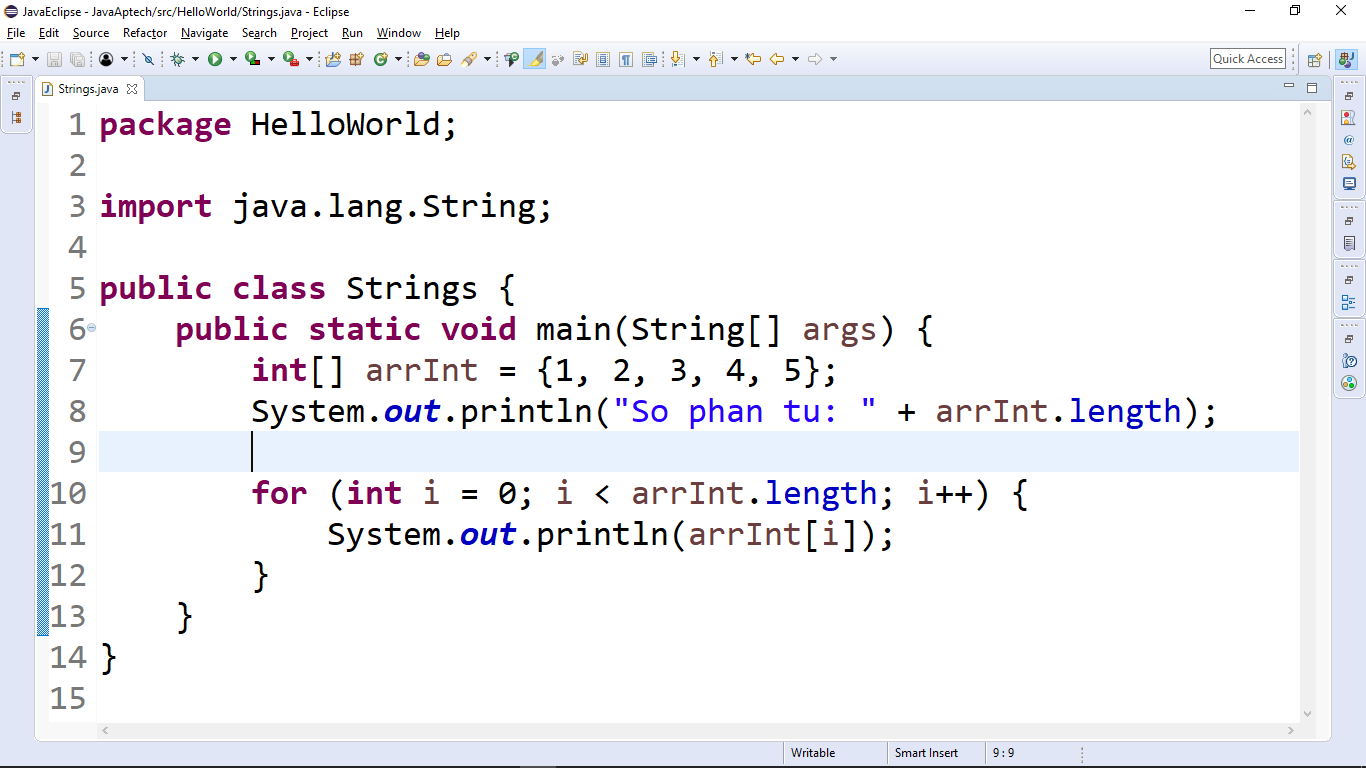
* Khởi tạo giá trị cho 1 số phần tử đầu mảng



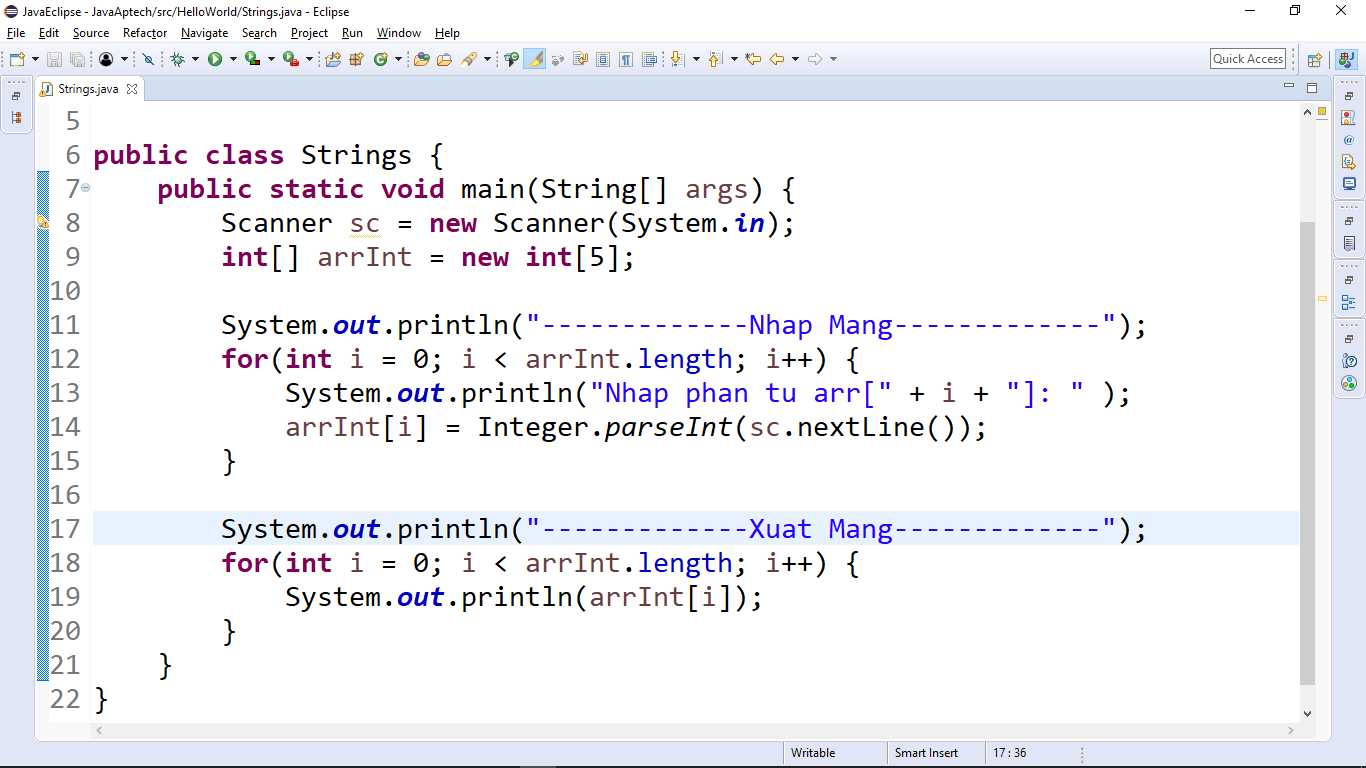
* Khởi tạo giá trị 0 cho mọi phần tử trong mảng



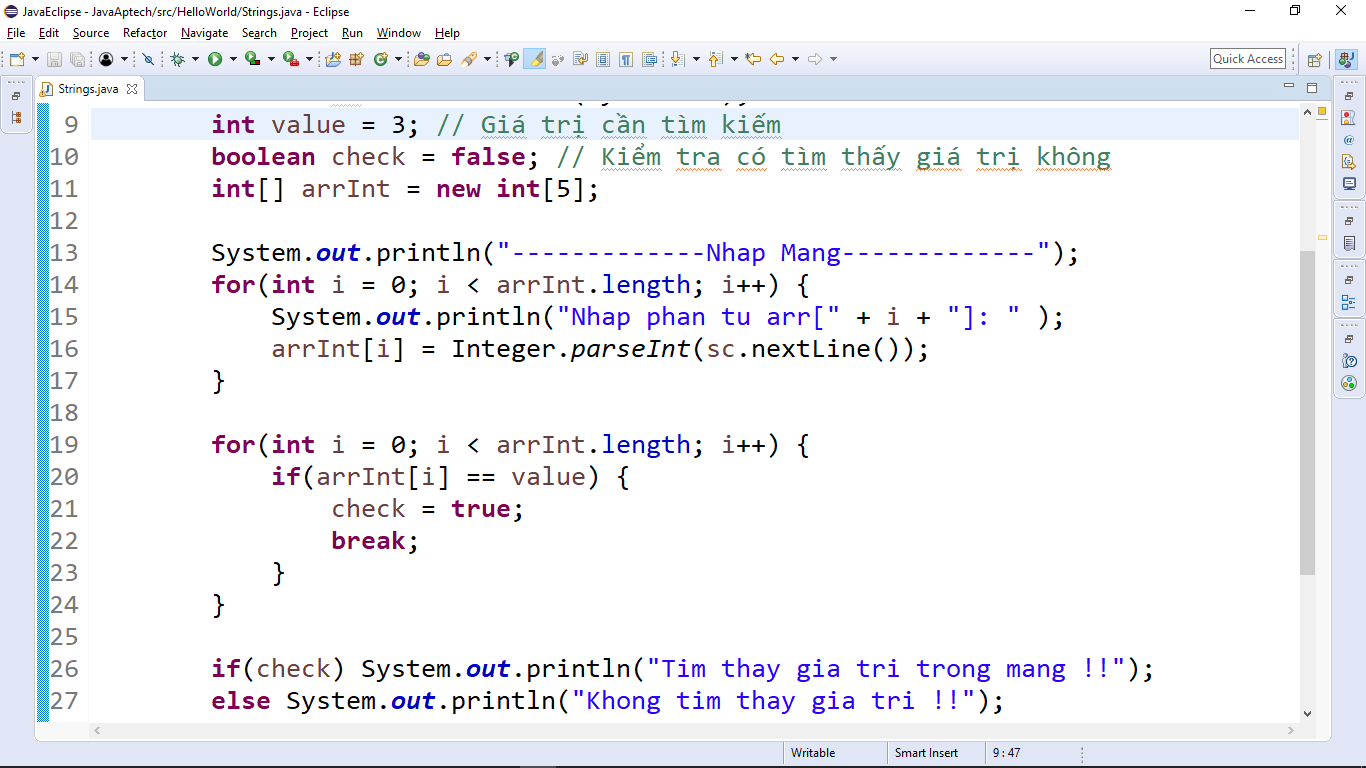
* Tự động xác định số phần tử có trong mảng



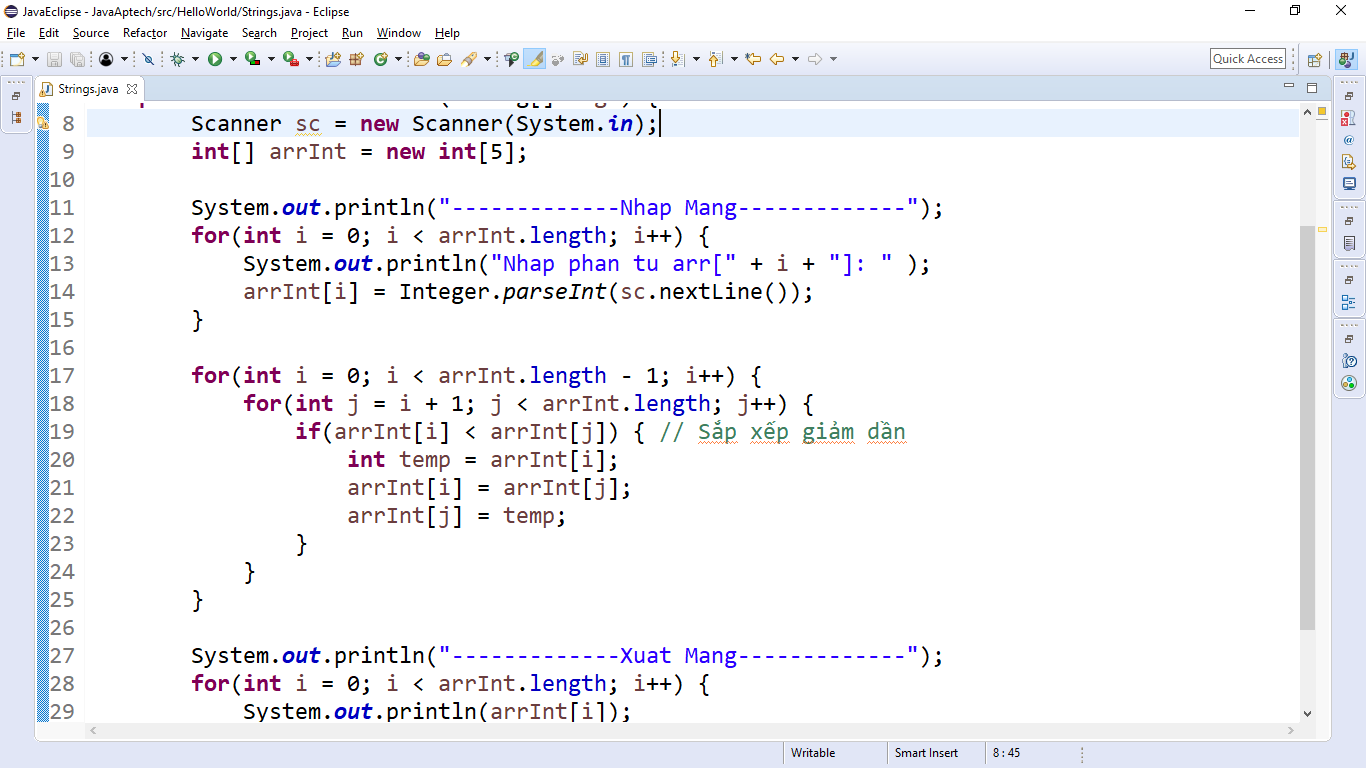
* + 1. **Các kỹ năng cơ bản được sử dụng trong mảng**
       1. Nhập. xuất mảng phần tử trong mảng



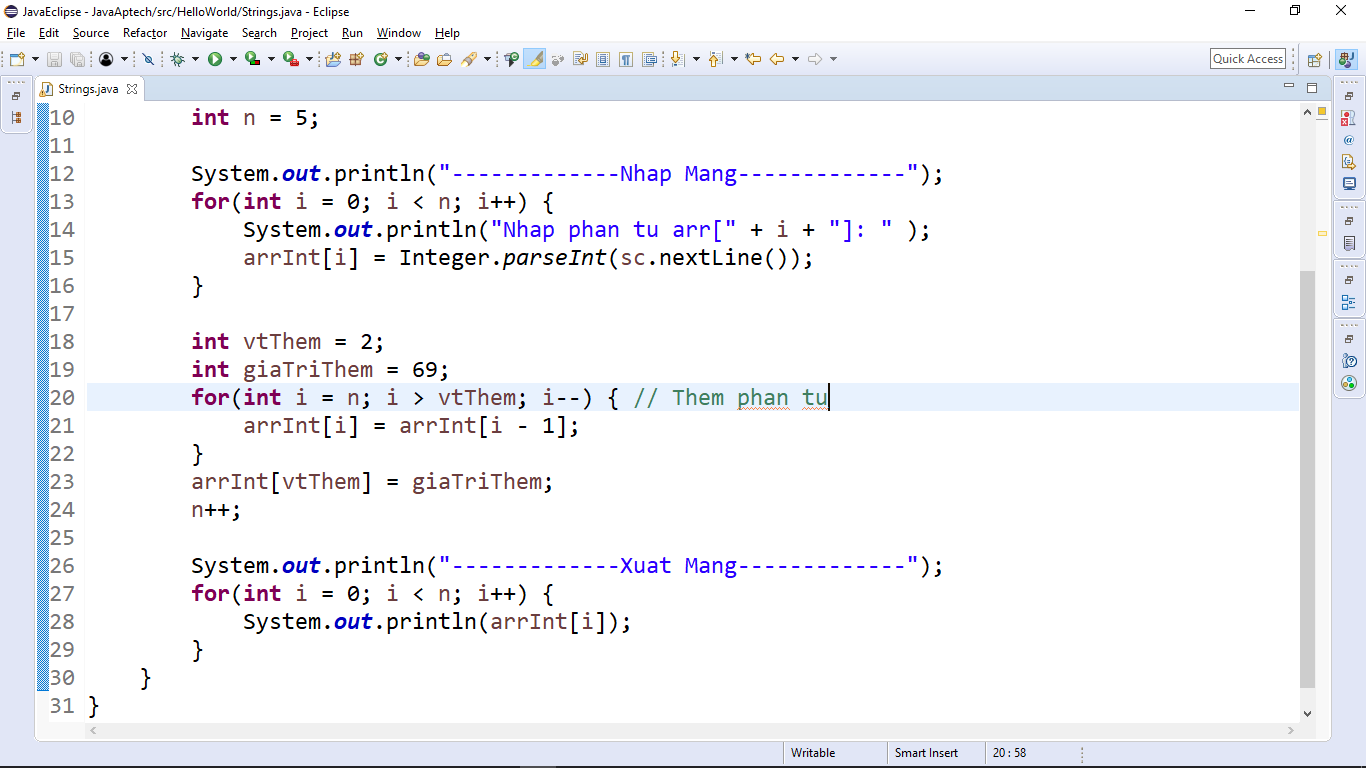
* + - 1. Tìm kiếm phần tử trong mảng
* Yêu cầu tìm kiếm phần tử chứa giá trị cần tìm trong mảng
* Ý tưởng:
* Dùng 1 biến để kiểm tra giá trị cần tim có trong mảng không



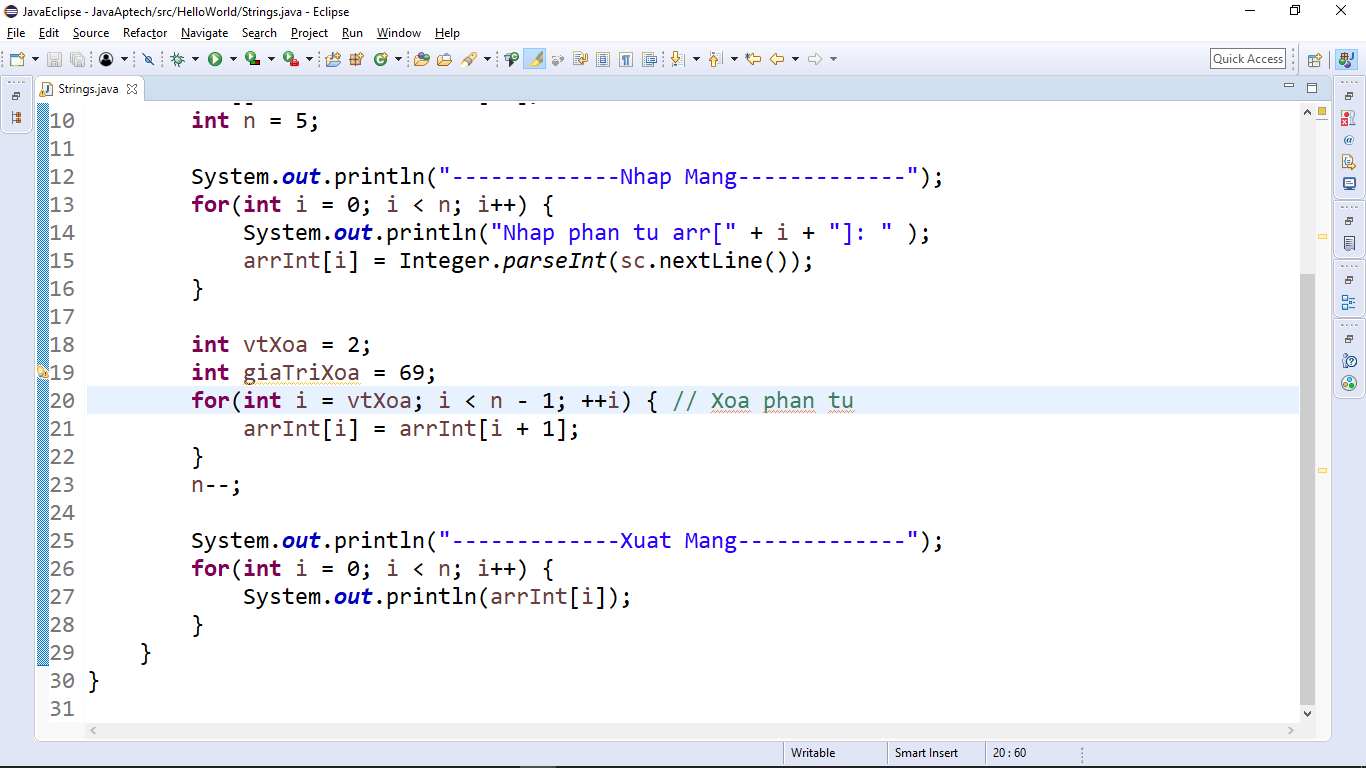
* + - 1. Sắp xếp phần tử trong mảng
* Yêu cầu sắp xếp các phần tử trong mảng giảm dần
* Ý tưởng:
* Sử dụng 2 biến so sánh 2 phần tử trong mảng và hoán vị.



* + - 1. Thêm phần tử trong mảng
* Yêu cầu thêm 1 phần tử bất kì trong mảng
* Ý tưởng:
* “Đẩy” các phần tử kể từ vị trí thêm sang bên phải qua 1 đơn vị
* Thay đổi giá trị ở vị trí thêm.
* Tăng số lượng phần tử xuống 1 đơn vị



* + - 1. Xóa phần tử trong mảng
* Yêu cầu xóa 1 phần tử bất kì trong mảng
* Ý tưởng:
* “Kéo” các phần tử kể từ vị trí xóa sang bên phải qua 1 đơn vị
* Giảm số lượng phần tử xuống 1 đơn vị



* 1. **Làm quen với mảng 2 chiều**

Tương tự như mảng 1 chiều.

* <Kiểu dữ liệu> [][] <tên biến mảng> = new <Kiểu dữ liệu> [<số phần tử>] hoặc

<Kiểu dữ liệu> <tên biến mảng>[][] = new <Kiểu dữ liệu> [<số phần tử>].

* 1. **Tìm hiểu về Arraylist**
* Tương tự như mảng 1 chiều nhưng khác 1 chút là mảng động, không cần phải khai báo số lượng phần tử ngay ban đầu. Mảng tự động co giãn vừa đủ sso lượng nhập vào.
  + 1. Một số phương thức hay dùng:
       1. **Add()**: Thêm phần vào mảng
       2. **Remove()**: Gỡ bỏ phần tử tại index đã cho. Ném IndexOutOfBoundsException nếu index ở ngoài dãy (index < 0 || index >= size())Size(): Trả về số phần tử trong list này
       3. **removeRanger()**: Gỡ bỏ từ list này tất cả phần tử mà có index ở giữa fromIndex và toIndex
       4. **Get()**: Trả về phần tử tại index đã cho. Ném IndexOutOfBoundsException nếu index đã cho là ở bên ngoài dãy (index < 0 || index >= size())
       5. **IndexOf()**: Trả về index trong list này của sự xuất hiện đầu tiên của phần tử đã cho, hoặc -1 nếu List không chứa phần tử này
       6. **lastIndexOf()**: Trả về index trong list này của sự xuất hiện cuối của phần tử đã cho, hoặc -1 nếu List không chứa phần tử này
       7. **clear()**: Gỡ bỏ tất cả phần tử từ list này

