**Buổi 2. Làm quen với CLASS và OBJECT**

A.Tổng quan về Class và Object.

1. Lập trình hướng đối tượng (OOP) là gì ?

- OOP ( Viết tắt của Object Oriented Progamming ) - Lập trình hướng đối tượng là một phương pháp lập trình dựa trên khái niệm lớp và đối tượng. OOP tập trung hơn vào cái đối tượng hơn là khai thác logic để thao tác chúng, giúp code dẽ quản lý, tái sử dụng được và dễ bảo trì

- Trừu tượng trong OOP: tổng hợp các tính chất của đối tượng và cho vô trong Class.



VD: Chương trình trên tạo ra một đối tượng học sinh, sau đó thực hiện gán tên và tuổi cho đối tượng này và hiển thị thông tin của đối tượng ra màn hình.

2. Class và Object.

a.Đối tượng (Object): Là 1 thể hiện có thuộc tính và hành vi.

Đối tượng trong OOP gồm 2 thành phần chính:

* Thuộc tính (Attribute): là những thông tin đặc điểm đối tượng
* Phương thức (Method): là những hành vi mà đối tượng có thể thực hiện

b.Lớp (Class):

Là sự trừu tượng hóa của đối tượng. Là khuôn mẫu của một tập hợp các đối tượng có cùng thuộc tính (attributes) và hành vi (methods). Cấu trúc của 1 lớp gồm:

* Phạm vi truy cập (access modifier): phạm vi truy cập của lớp, của thuộc tính và phương thức. (chẳng hạn: private, protected, default, public)
* Tên lớp (class name): Mỗi lớp đều có một tên riêng biệt, để phân biệt với các lớp khác trong cùng một phạm vi.
* Các thuộc tính (attributes): Mô tả các trường để lưu dữ liệu cho mỗi đối tượng mà lớp đang định nghĩa hay lưu các tham chiếu đến đối tượng của lớp khác.
* Các phương thức (methods): mỗi phương thức của lớp thực chất là một hàm được viết riêng cho các đối tượng của lớp, và được gọi đến để tác động lên các đối tượng của lớp đó.

VD : Class: bản thiết kế, Object: Ngôi nhà. Khác nhau là ngôi nhà là thực thể có thật, tạo ra từ Class.

3. Phạm vi truy cập (Access Modifier): Gồm 4 phạm vi truy cập cho lớp.

OOP Java gồm 4 phạm vi truy cập cho lớp, thuộc tính, phương thức, gồm: private, protected, default, public.

* Phạm vi private: chỉ cho phép truy cập nội bộ trong class.
* Phạm vi protected: là một loại phạm vi cho phép truy cập được từ trong hay cả ngoài package (gói), nếu là ngoài package thì phải thông qua tính kế thừa (inheritance). Cả phạm vi private và protected đều chỉ áp dụng bên trong class như thuộc tính, phương thức, ... Không thể áp dụng cho phạm vi truy cập của lớp hoặc interface.
* Phạm vi default: là phạm vi mặc định, khi khai báo không cần ghi gì cả ,đối với phạm vi default này, thì phạm vi truy cập cho phép chỉ trong cùng package (gói).

VD: int age

* Phạm vi public: là phạm vi rộng nhất, bất cứ chỗ nào trong project đều có thể truy cập đến được. VD : public class A{} thì bất cứ đâu cũng có thể truy cập vào class.

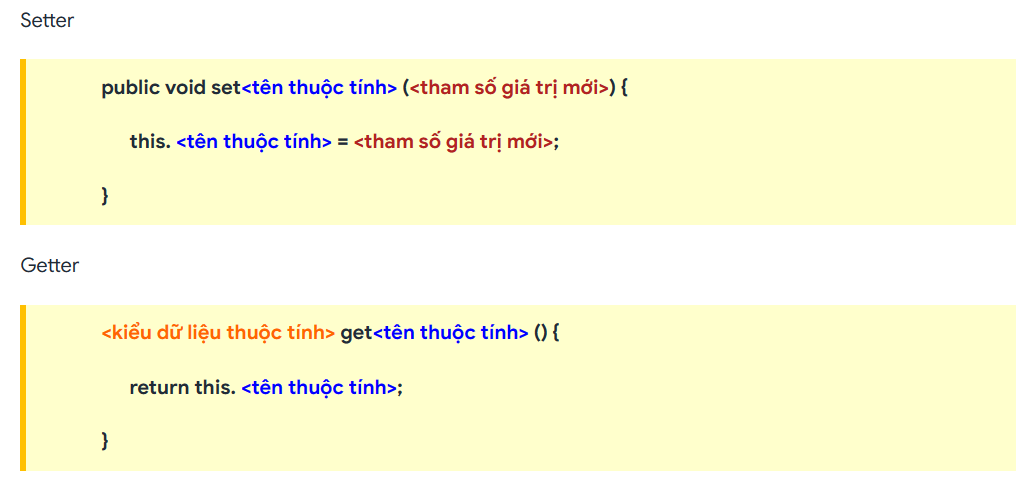


Nên để All Private (trừ những cái cần thiết) để không bị hack 😊. Tuy nhiên bài ở trg thì public để dễ truy cập, ko thì lại them Getter, Setter .

4. Getter và Setter trong Java:

- Là 2 phương thức để cập nhật hoặc lấy ra giá trị của thuộc tính, đặc biệt với thuộc tính dạng private.

- Cú pháp:



* Việc ta sử dụng Getter, Setter giúp ta dễ dàng kiểm soát các thuộc tính.

VD : ta có 1 thuộc tính là tuổi, ta có thể kiểm soát nó trong giới hạn (0 <= Age <= 100) nhờ Setter.

* Nên sử dụng cho các thuộc tính ở kiểu “private” vì ta muốn kiểm soát thuộc tính thì không nên cho nó có thể dễ dàng truy cập theo cách khác mà chỉ sử dụng Getter, Setter để kiểm soát thôi.



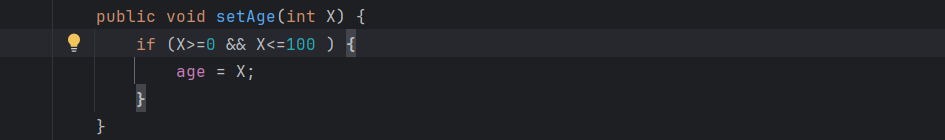
Xuất: 

Lý do cả 2 lần đều xuất ra 22 vì phương thức setAge() đã kiểm soát và không cho gán giá trị -5 vào Age.

5. Con trỏ this:

Trong ví dụ trên, ta có thể thấy 1 dòng là this.age = age.Lý do có cụm ‘This’ vì:

* Khi biến cục bộ hoặc tham số trong 1 phương thức có cùng tên với 1 thuộc tính trong lớp. Và chương trình sẽ ưu tiến biến cục bộ.VD: Nếu ta sử dụng age = age thì chương trình sẽ xuất ra null thay vì số tuổi ta nhập.
* Cách xử lý : + C1: thay tên biến cục bộ khác với tên thuộc tính:



+ C2: sử dụng con trỏ this: Tác dụng: làm code dễ đọc hơn, dễ quản lý hơn (Vì sẽ dễ quản lý các giá trị truyền vào).

* Lưu ý: Không dung từ khóa this cho đối tượng static vì nó thuộc về lớp, còn this sử dụng cho đối tượng.

6.Từ khóa Static.

- **Từ khóa static trong Java** được sử dụng chính để quản lý bộ nhớ. Chúng ta có thể áp dụng từ khóa static với các biến, các phương thức, các khối, các lớp lồng nhau(nested class). Từ khóa static thuộc về lớp chứ không thuộc về instance(thể hiện) của lớp => Có thể truy cập thuộc tính, phương thức static từ bất cứ đâu trong chương trình mà không cần khởi tạo đối tượng mới.

- VD:

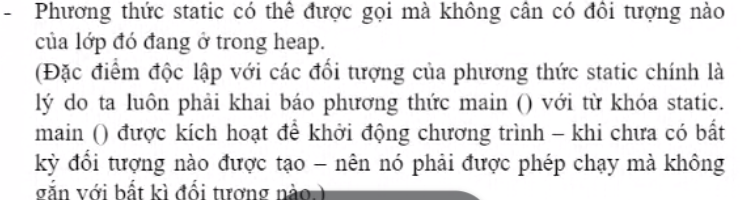


Xuất : 

* Có thể truy cập “dem” bằng tên của class và cả tên của đối tượng (Tuy nhiên không nên truy cập bằng tên của đối tượng vì nó không phải là cách thông thường và có thể sinh ra lỗi không đáng có).
* Khi ta tạo ra các đối tượng, thì “dem” cũng tăng lên, và truy cập từ cả tên class và tên đối tượng đều ra kết quả như nhau => Giá trị này được chia sẻ với tất cả đối tượng tạo ra trong lớp.

+) Phương thức static: Tương tự như thuộc tính static.

* Lưu ý: + Có thể truy cập thuộc tính static từ phương thức static.Tuy nhiên không được truy cập thuộc tính không static từ phương thức static. Vì: Thuộc tính không static không được chia sẻ giữa các đối tượng, mà thuộc về riêng nó.
* # :



B. String và Stringbuilder trong Java.

1. String:

- String trong Java rất đặc biệt: Nó không chỉ là 1 chuỗi kí tự mà còn là 1 lớp.

- String là bất biến, hằng, không thay đổi được.

- Các phương thức:

1. Độ dài của xâu: s.length();

2.Truy cập vào 1 phần tử của xâu: s.charAt(k);

3.Tìm vị trí xuất hiện của xâu con s2 trong s1: int ID = s1.indexOf(s2)

4.Chuyển xâu kí tự sang in hoa, in thường.

S = s.toUpperCase();

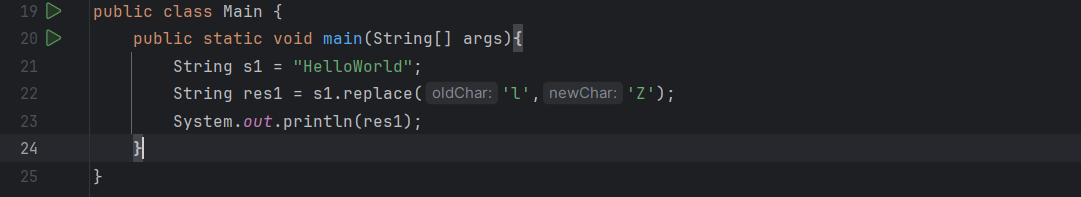
S = s.toLowerCase();

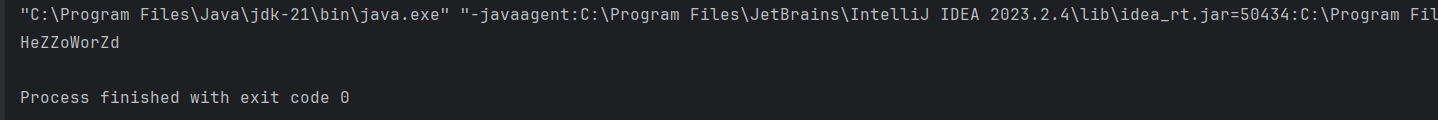
5.Thay thế ký tự (xâu con trong S1 = S2)

S1.replace(c,d);

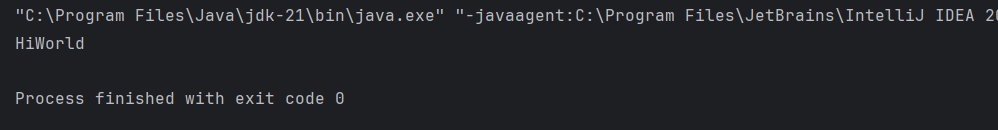
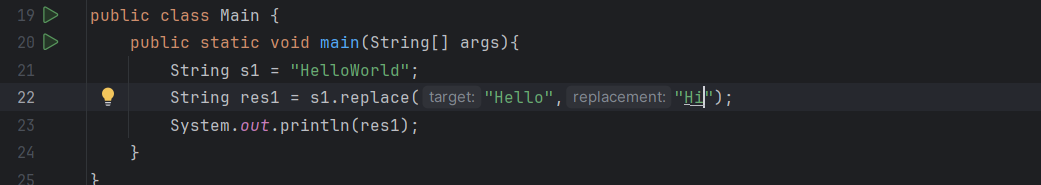
S2.replace(s1,s3)

VD1:





VD2:



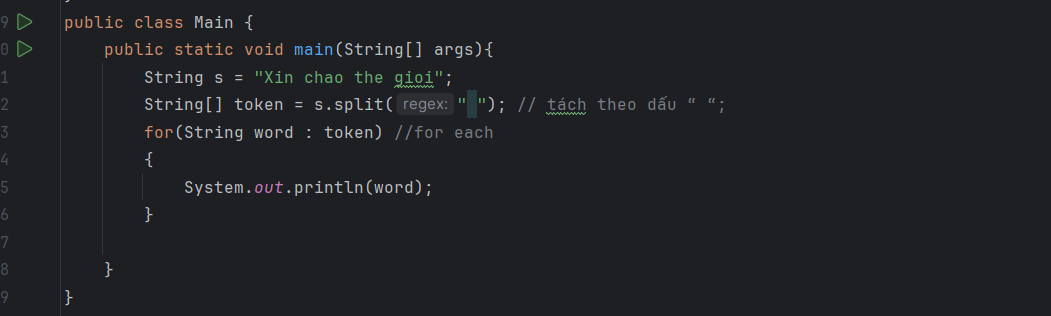
* Lưu ý: với kí tự thì không được là kí tự rỗng ‘’, với xâu rỗng thì được, lúc ấy thi sẽ xóa đi kí tự được chọn.

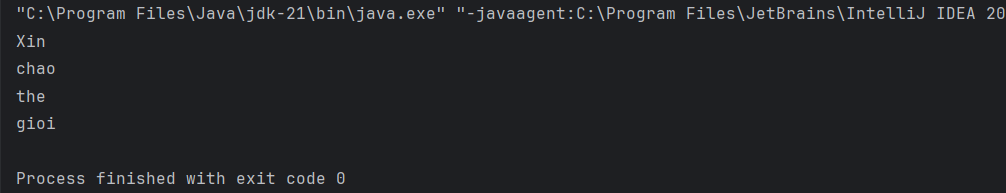
6. Kiểm tra xem xâu s1 có bắt đầu/Kết thúc bởi xâu s2 hay không:

Boolean b = s1.startsWith(s2);

Boolean b = s1.endsWith(s2);

7.Tách 1 xâu thành mảng các xâu theo phương thức cho trước:



Xuất: 

8. Tách 1 xâu ra từ vị trí x -> y:

String tmp = s.substring(x,y);

9. So sánh 2 xâu:

A.compareTo(B);

=> Trả về > 0 nếu A > B, trả về = 0 nếu A = B, trả về < 0 nếu A < B;

2. Stringbuilder:

- StringBuilder Class làm cho String class trở nên linh động hơn vì nó có thể tạo ra một set của các chuỗi kí tự có thể thay đối tượng. StringBuilder Class cũng có một loạt các methods (phương thức) để tương tác hay tác động lên các String objects như trong String Class



- StringBuilder cung cấp những methods(phương thức) mà String Class không hề có, ví dụ như phương thức insert(), delete() và reverse(). Khi cần thao tác nhiều hay chuỗi quá dài thì chúng ta nên sử dụng StringBuilder bởi vì chúng nhanh hơn và tiết kiệm bộ nhớ hơn rất nhiều so với String.

C. Phương thức mặc định của Class Object: equals(), hashCode().

1. Phương thức equals() và phương thức hashCode(): là những phương thức rất quan trọng đối với tất cả các đối tượng trong Java.

- Mặc định, các phương thức này được định nghĩa trong đối tượng Object, là đối tượng cha của tất cả các đối tượng trong Java. Nhưng để sử dụng đúng mục đích của 2 phương thức này, chúng ta cần override lại chúng trong các đối tượng

a. Phương thức equals:

**Phương thức equals()**

Phương thức này dùng để kiểm tra 2 đối tượng bất kỳ có bằng nhau hay không?

Trong Java có 2 kiểu dữ kiệu:

|  |  |
| --- | --- |
| Primitivate: | Object |
| Có thể sử dụng toán tử “==” để so sánh | Bắt buộc phải override lại phương thức equals() trong đối tượng Object để so sánh xem 2 đối tượng cùng class có bằng nhau hay không? |
|  | Không ghi đè:     * Dù giống nhau, ct vẫn coi là 2 đt riêng biệt.   Ghi đè:     * Lúc này mới trả về True |

### b. Phương thức hashCode()

Các đối tượng như HashMap, HashTable sử dụng các đối tượng key để tìm kiếm các đối tượng value tương ứng. Và để tối ưu cho việc tìm kiếm, Java sử dụng các hash value lấy từ phương thức hashCode() của các đối tượng key để gom nhóm các đối tượng này trong những cái gọi là các bucket. Mỗi bucket tương ứng với một giá trị hash value.

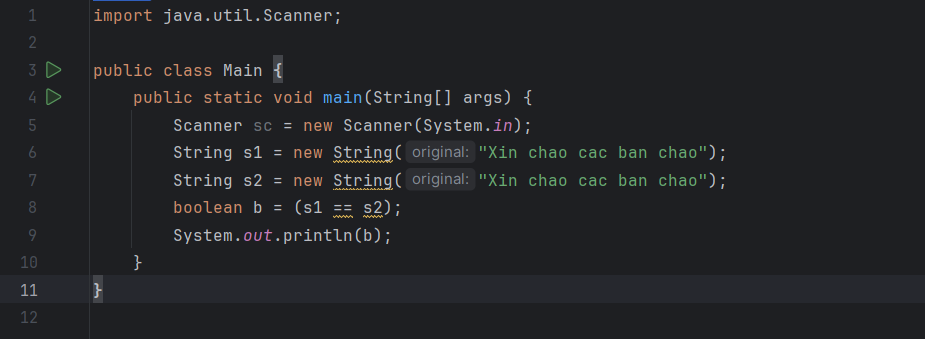
Khi tìm kiếm một đối tượng bất kỳ trong HashMap hay HashTable, đầu tiên Java sẽ lấy hash value của đối tượng key mà chúng ta muốn tìm kiếm. Sau đó nó sẽ kiểm tra các bucket để xem giá trị hash value đó có trùng khớp với bucket nào hay không. Nếu thoả mãn một bucket nào đó thì Java sẽ trả về giá trị của đối tượng value mà chúng ta muốn tìm kiếm.

Nhiều đối tượng có thể có cùng hash value, và do đó chúng có thể nằm chung một bucket. Trong trường hợp này, Java sẽ sử dụng tiếp phương thức equals() để tìm chính xác đối tượng value mà chúng ta muốn tìm kiếm.

Ở đây chúng ta có một khái niệm gọi là contract giữa phương thức equals() và phương thức hashCode() đó là:

* Nếu hai đối tượng là bằng nhau thì chúng phải có cùng hashCode().
* Nếu hai đối tượng có cùng hashCode() thì chúng có thể bằng nhau hoặc là không.

c.Toán tử ‘==’ : - Với biến tham chiếu (String,Arr) :Kiểm tra xem 2 biến Chuỗi có cùng tham chiếu tới 1 địa chỉ hay ko, chứ không phải so sánh giá trị của xâu.



Lưu ý: tạo string = new thì mới khác địa chỉ ( trong bộ nhớ Heap)

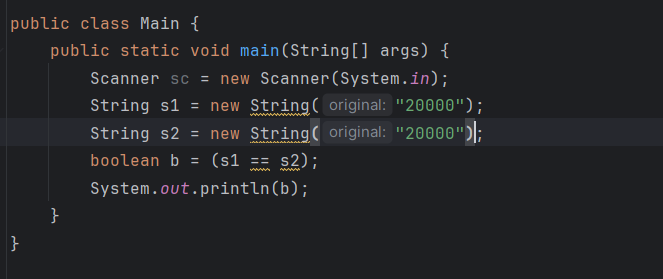
Các kiểu nguyên thủy int, float, … thì tương tự C++

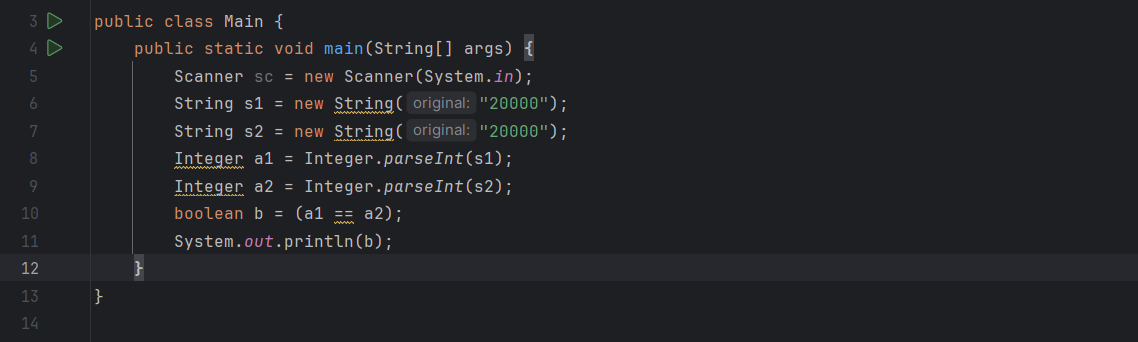
3. Package:

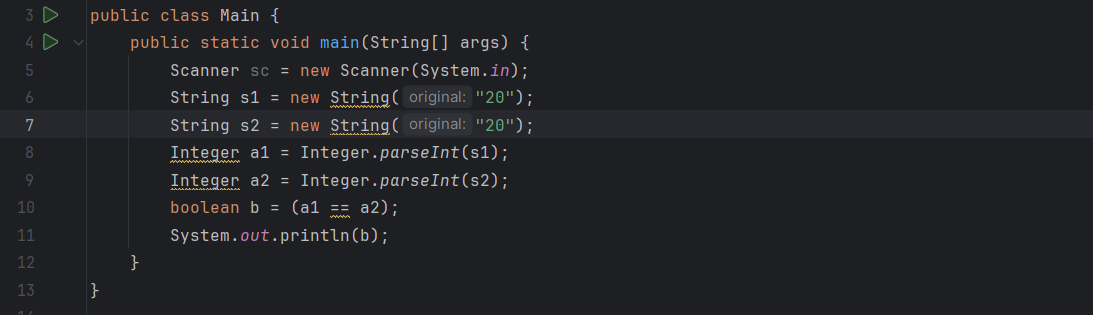
4. Heap và Stack:

a. Heap: Lưu trữ đối tượng tạo ra bới New

- Trong heap tồn tại 1 pool cho các số từ -127 -> 127. Các số bé thì đc tham chiếu vô đó. Còn các số lớn phải tạo riêng. ==🡺 nếu so sánh == số bé thì true, số lớn thì false.







Để fix => sd equals.

b. Stack: lưu biến nguyên thủy. ( int ,float…)