# [Buổi 2] LÀM QUEN VỚI CLASS VÀ OBJECT

## Tổng quan về class và object;

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

- Là một phương pháp để giải quyết bài toán lập trình mà khi áp dụng code sẽ trở nên dễ phát triển và dễ bảo trì hơn. Đây là một phương pháp tiên tiến, phù hợp để phát triển hầu hết các ứng dụng hiện nay. Phương pháp này sẽ giải quyết bài toán bằng cách chia chương trình thành các đối tượng, với mỗi đối tượng sẽ có các thuộc tính (dữ liệu) và hành vi riêng (phương thức).

### 2. Class và Object là gì ? Cách phân biệt chúng ? Con trỏ this ?

- Lớp (class) là nơi định nghĩa thông tin về các đối tượng, cũng có thể hiểu lớp là một kiểu dữ liệu người dùng tự định nghĩa mà biến của kiểu dữ liệu này được coi như là một đối tượng.

- Đối tượng (Object) là một thể hiện (instance) của lớp, có thể hiểu đối tượng là biến của class mà ta ví dụ như là một kiểu dữ liệu.

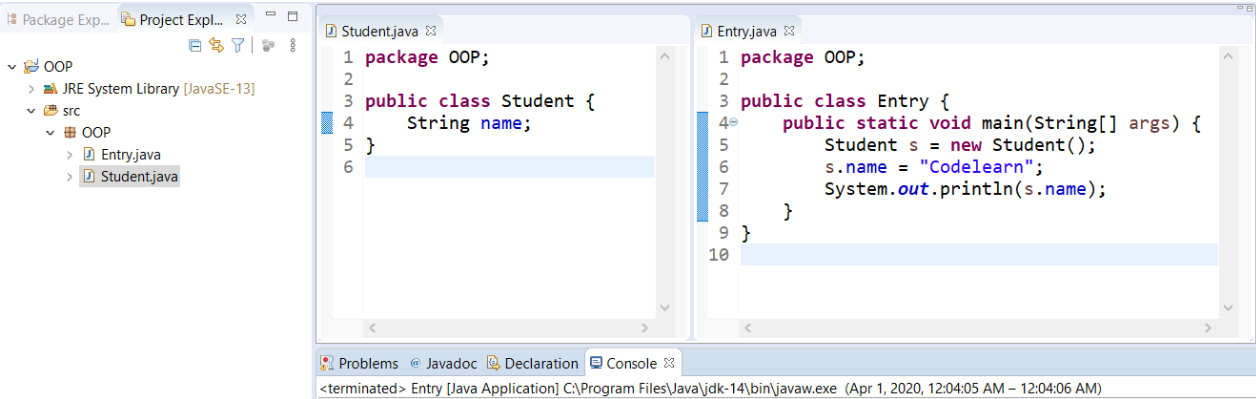
- Con trỏ this: dùng để truy cập các thuộc tính hay phương thức của lớp hiện tại, khi có biến trùng tên với thuộc tính của lớp ta dùng con trỏ this để phân biệt được đâu mới là thuộc tính của lớp.

### 3. Access modifier trong Java:

- Là khái niệm để xác định xem một biến hay một phương thức có được truy xuất từ bên ngoài hay không. Trong Java có 4 loại phạm vi truy cập là private, default, protected, public.

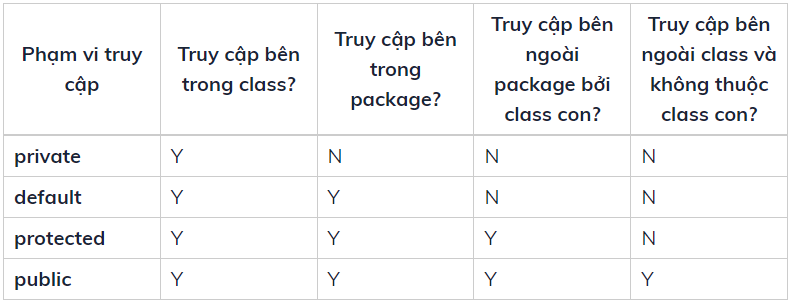
- Phạm vi truy cập private: là phạm vi truy cập hạn chế nhất, tất cả các thuộc tính và phương thức sẽ chỉ được truy xuất bên trong lớp đó.

- Phạm vi truy cập defaut; Khi khai báo một thuộc tính hoặc một phương thức mà không khai báo phạm vi truy cập thì chương trình sẽ hiểu là bạn đang khai báo với phạm vi truy cập mặc đỉnh chỉ có thể được truy xuất trong cùng một package.



- Phạm vi truy cập protected: là phạm vi truy cập tương tự với default nhưng ít hạn chế hơn ở chỗ các phương thức có phạm vi truy cập này có thể được truy xuất từ một lớp con của package khác.

- Phạm vi truy cập public:là phạm vi truy cập rộng nhất, các thuộc tính và phương thức có phạm vi truy cập này có thể được truy xuất ở bất cứ đâu (kể cả từ package khác).



### 4. Getter và setter:

- Để đảm bảo tính đóng gói, các thuộc tính nên để phạm vi truy cập là private và ta chỉ có thể truy xuất chúng qua các phương thức public (gọi là các setter, getter).

- Getter là phương thức với yêu cầu trả về là giá trị của thuộc tính.

- Setter là phương thức để cài đặt lại các giá trị của thuộc tính private.

class Student {

    private String name;

    private *int* age;

    private *double* gpa;

    public *String* getName() {

        return name;

    }

    public *void* setName(*String* *name*) {

*this*.name = name;

    }

    public *int* getAge() {

        return age;

    }

    public *void* setAge(*int* *age*) {

*this*.age = age;

    }

    public *double* getGpa() {

        return gpa;

    }

## Từ khóa static:

- Biến static: là biến có thể dùng mà không cần phải khởi tạo đối tượng và được gọi đến bằng tên của Class chứ không thể gọi bằng tên Object.

- Ngoài việc sử dụng mà không cần phải khởi tạo đối tượng thì biến static còn có đặc điểm là được chia sẻ bởi tất cả các đối tượng trong chương trình (giá trị của biến static trong tất cả các chương trình là giống nhau). Chính vì tính chất này nên biến static thường được dùng để lưu thông tin chung cho tất cả các đối tượng và lưu các hằng số.

Lưu ý: Biến được khai báo với thuộc tính static không được coi là thuộc tính do nó không thuộc đối tượng cụ thể nào.

class Counter {

    static *int* count;

    public Counter() {

        count++;

        System.out.println(count);

    }

}

class Entry {

    public static *void* main(String[] *args*) {

        Counter c1 = **new** Counter();

        Counter c2 = **new** Counter();

        Counter c3 = **new** Counter();

    }

}

Kết quả khi chạy chương trình:

1

2

3

- Phương thức static: tương tự biến static, được khai báo với từ khóa static mà không cần tạo ra đối tượng. Trong cùng một lớp, phương thức static chỉ có thể gọi tới phương thức và thuộc tính static khác, không thể gọi tới phương thức, thuộc tính không phải private.

## String và StringBuilder:

### 1. String:

- CharSequence Interface: được sử dụng để biểu diễn chuỗi các ký tự. Nó được cài đặt trong lớp String, StringBuffer và StringBuilder nên từ đó ta có 3 cách để tạo ẻa chuỗi ký tự.

- String là bất biến (immutable) có nghĩa là khi nào bạn thay đỏi giá trị của chuỗi bất kỳ nào thì một instance mới được tạo ra. Đối với chuỗi có thể thay đổi, ta nên sử dụng các lớp String Buffer và StringBuilder.

- Cách khai báo: sử dụng string literal hoặc từ khóa new.

- Các phương thức phổ biến trong Java:

|  |  |  |
| --- | --- | --- |
| Kiểu trả vể | Tên phương thức | Mô tả |
| char | charAt(int index) | Trả về kí tự char ở index |
| int | length() | Trả về độ dài chuỗi |
| String | (static) format (String format, …) | Trả về chuỗi được format |
| String | Substring(int beginIndex) | Trả về chuỗi con bắt đầu từ vị trí index |
| Boolean | contains(CharSequence s) | Kiểm tra chuỗi có chứa chuỗi không |
| String | (static) join(CharSequence 1, …) | Trả về chuỗi được nối từ nhiều chuỗi |
| Boolean | equals(Object) | Trả về sự tương đương của chuỗi với đối tượng |
| String [] | Split(String regex) | Trả về các chuỗi được tách theo giá trị regex. |
| String | replace(char old, char new) | Thay thế tất cả các kí tự cũ bằng kí tự mới. |
| Int | indexOf(String substring) | Trả về index của chuỗi con |
| String | toUperCase() | Trả về chuỗi chữ hoa |
| String | toLowerCase() | Trả về chuỗi chữ thường |

- Trong Java, đối tượng bất biến (immutable) không thể thay đổi:

public class Testimmutablestring {

 public static void main(String args[]) {

  String s = "Hello";

  s.concat(" Java");//phương thức concat() để nối thêm chuỗi vào đuôi chuỗi s.

  System.out.println(s);//sẽ chỉ in ra "Hello" vì các chuỗi này là đối tượng không thể thay đổi.

 }

}

- Tuy nhiên, nếu gắn s vào biến tham chiếu, s sẽ được tham chiếu đến chuỗi mới. Lúc này, s sẽ trỏ đến chuỗi mới nhưng chuỗi cũ vẫn tồn tại trong Pool hằng chuỗi và không bị thay đổi.

public class Testimmutablestring {

 public static void main(String args[]) {

  String s = "Hello";

  s = s.concat(" Java");

  System.out.println(s);

 }

}

- So sánh chuỗi trong Java (3 cách):

+ sử dụng phương thức equals():

* equals(Object another): so sánh có phân biệt chữ hoa, chữ thường.
* equalsIgnoreCase(String another): không phân biệt chữ hoa, chữ thường.

+ sử dụng toán tử ==: dùng để so sánh tham chiếu chứ không phải giá trị của 2 chuỗi.

+ sử dụng phương thức compareTo(): so sánh theo thứ tự từ điển tăng dần, trả về

* 0 (s1==s2)
* >0 (s1>s2)
* <0 (s1 < s2)

- Nối chuỗi trong Java (2 cách):

+ sử dụng toán tử +: được thực hiện qua lớp StringBuilder (hoặc StringBuffer) và phương thức append() của nó.

+ sử dụng phương thức concat()

- Substring trong Java, có 2 cách để lấy chuỗi con từ một chuỗi đã cho:

+ substring(int startIndex): trả về chuỗi con tính từ index bắt đầu.

+ substring(int startIndex, int endIndex): trả về chuỗi trong khoảng index đầu và cuối.

- Những phương thức quan trọng của lớp String:

+ toUperCase() và toLowerCase().

+ trim(): xóa khoảng đầu và khoảng trắng của chuỗi.

+ strartsWith(String start) và endsWith(String end): kiểm tra chuỗi có bắt đầu hoặc kết thúc bằng chuỗi start hay end không.

+ charAt(int index): trả về kí tự tại vị trí Index.

+ length(): trả về độ dài chuỗi

+ intern(): nếu Pool chứa một chuỗi bằng đối tượng chuỗi này (so sánh bằng phương thức equals(Object) thì trả về chuỗi trong Pool. Ngược lại, đói tượng chuỗi sẽ được thêm vào Pool và trả về tham chiếu của đối tượng chuỗi này.

String s=**new** String("Hello");

String s2=s.intern();

System.out.println(s2);//Hello

Output:

Hello

+ valueOf(): chuyển đổi các kiểu dữ liệu khác thành kiểu String.

int a=10;

String s=String.valueOf(a);

System.out.println(s+10);

Output:

1010

### 2. StringBuilder trong Java:

- Trong Java, lớp StringBuilder được sử dụng để tạo chuỗi có thể thay đổi (muitable). Lớp StringBuilder tương tự như lớp StringBuffer ngoại trừ nó không đồng bộ (non-synchronized).

- Các Constructor quan trong của lớp StringBuilder trong Java:

+ StringBuilder(): tạo ra một Builder chuỗi với dung lượng ban đầu là 16.

+ StringBuilder (String str): tạo ra một Builder chuỗi với chuỗi cụ thể.

+ StringBuilder(int capacity): tạo ra một Builder chuỗi với dung lượng được chỉ định như độ dài chuỗi.

- Các phương thức của lớp StringBuilder trong Java:

**1) public StringBuilder append(String s):** dùng để nối các chuỗi được chỉ định với chuỗi hiện tại, có thể nối với các kiểu dữ liệu khác như boolean, int, …

public class StringBuilderExam1 {

 public static void main(String args[]) {

  StringBuilder sb = **new** StringBuilder("Hello ");

  sb.append("Java");//đến đây chuỗi ban đầu đã bị thay đổi

  System.out.println(sb);//in Hello Java

 }

}

**2) public StringBuilder insert(int offset, String s):** được sử dụng để chèn chuỗi chỉ định với chuỗi hiện tại ở vị trí offset, có thể nối với các kiểu dữ liệu khác như boolean, int, char, double, …

public class StringBuilderExam2 {

 public static void main(String args[]) {

  StringBuilder sb = **new** StringBuilder("Hello ");

  sb.insert(1, "Java");//đến đây chuỗi ban đầu đã bị thay đổi

  System.out.println(sb);//in -> HJavaello

 }

}

**3) public StringBuilder replace(int startIndex, int endIndex, String str):** dùng để thay thế chuỗi từ vị trí startIndex đến vị trí endIndex bằng chuỗi str.

public class StringBuilderExam3 {

 public static void main(String args[]) {

  StringBuilder sb = **new** StringBuilder("Hello");

  sb.replace(1, 3, "Java");

  System.out.println(sb);//in -> HJavalo

 }

}

**4) public StringBuilder delete(ints startIndex, int endIndex):** xóa chuỗi từ vị trí startIndex đến endIndex.

public class StringBuilderExam4 {

 public static void main(String args[]) {

  StringBuilder sb = **new** StringBuilder("Hello");

  sb.delete(1, 3);

  System.out.println(sb);//in -> Hlo

 }

}

**5) public StringBuilder reverse():** đảo ngược chuỗi hiện tại.

public class StringBuilderExam5 {

 public static void main(String args[]) {

  StringBuilder sb = **new** StringBuilder("Hello");

  sb.reverse();

  System.out.println(sb);//in -> olleH

 }

}

**6) public int capacity():** trả về dung lượng hiện tại.

public class StringBuilderExam6 {

 public static void main(String args[]) {

  StringBuilder sb = **new** StringBuilder();

  System.out.println(sb.capacity());//mặc định là 16

  sb.append("Hello");

  System.out.println(sb.capacity());//đến đây vẫn là 16

  sb.append("java is my favourite language");

  System.out.println(sb.capacity());//đến đây là (16\*2)+2=34 i.e (dung lượng cũ\*2)+2

 }

}

**7) public void ensureCapacity(int minimumCapacity):** được sử dụng để đảm bảo dung lượng ít nhất bằng mức tối thiểu.

public class StringBuilderExam7 {

 public static void main(String args[]) {

  StringBuilder sb = **new** StringBuilder();

  System.out.println(sb.capacity());//mặc định là 16

  sb.append("Hello");

  System.out.println(sb.capacity());//đến đây là 16

  sb.append("java is my favourite language");

  System.out.println(sb.capacity());//đến đây là (16\*2)+2=34 i.e (dung lượng cũ\*2)+2

  sb.ensureCapacity(10);//đến đây không có sự thay đổi

  System.out.println(sb.capacity());//đến đây là 34

  sb.ensureCapacity(50);//đến đây là (34\*2)+2

  System.out.println(sb.capacity());//đến đây là 70

 }

}

**8) public char charAt(int index):** trả về ký tự tại vị trí index.

**9) public int length()**: trả về độ dài chuỗi hiện tại.

**10) public String substring(int beginIndex) / (int beginIndex, endIndex):** trả về chuỗi con với vị trí Index chỉ định.

### 2. StringBuffer trong Java:

- Lớp StringBuffer được sử dụng để tạo chuỗi có thể thay đổi (mutable). Lớp StringBuffer tương tự như lớp String ngoại trừ nó có thể thay đổi.

- Lớp Stringbuffer là thread-safe (luồng an toàn) có nghĩa là nhiều luồng (thread) không thể truy cập nó vào cùng thời điểm.

- Các Constructor của lớp StringBuffer:

+ StringBuffer(): tạo ra một bộ đệm chuỗi với dung lượng ban đầu là 16.

+ StringBuffer(String str): tạo ra một bộ đệm chuỗi với chuỗi cụ thể.

+ StringBuffer(int capacity): tạo ra một bộ đệm chuỗi với dung lượng chỉ định như độ dài chuỗi.

- Các phương thức của lớp StringBuffer:

**1) public synchronized StringBuffer append(String s):** nối thêm các chuỗi chỉ định vào chuỗi hiện tại, có thể nối các kiểu dữ liệu khác như boolean, int ,…

**2) public synchronized StringBuffer insert(int offset, String s):** được sử dụng để chèn chuỗi chỉ định vào chuỗi hiện tại ở vị trí offset, có thể chèn các kiểu dữ liệu khác như boolean, int, char, double,…

**3) public synchronized StringBuffer replace(int startIndex, int endIndex, String str):** chèn chuỗi chỉ định vào chuỗi hiện tại từ vị trí startIndex đến endIndex.

**4) public synchronized StringBuffer delete(int startIndex, int endIndex):** xóa chuỗi từ vị trí startIndex đến vị trí endIndex.

**5) public synchronized StringBuffer reverse():** đảo ngược chuỗi hiện tại.

**6) public int capacity():** trả về dung lượng hiện tại.

**7) public void ensureCapacity(int minimumCapacity):** được sử dụng để đảm bảo dung lượng ít nhất bằng mức tối thiểu nhất định.

**8) public char charAt(int index):** trả về kí tự tại vị trí index.

**9) public int length():** được sử dụng để trả về chiều dài của chuỗi.

**10) public String substring(int beginIndex) / (int beginIndex, int endIndex):** trả về chuỗi con ở vị trí bắt đầu và vị trí kết thúc được chỉ định.

## Tìm hiểu về toán tử == và 2 phương thức mặc định của Class Object: equals(), hashCode():

### Toán tử == :

- Dùng để so sánh 2 giá trị, cách hoạt động của toán tử == phụ thuộc vào loại dữ liệu của toán hạng:

+ khi sử dụng với các kiểu dữ liệu nguyên thủy (primitive data tyoes) như int, char, float, double, long, … thì toán tử == so sánh giá trị của 2 biến.

+ khi sử dụng với các đối tượng (objects), toán tử == so snash địa chỉ bộ nhớ mà các biến tham chiếu đến. Nó kiểm tra xem liệu 2 biến tham chiếu có trỏ đến cùng một đối tượng trong bộ nhớ hay không, vì vậy khi so sánh các đối tượng hoặc các chuỗi với nhau ta nên sử dụng phương thức equals() mà ta đã ghi đi phương thức.

### 2. Class Object và 2 phương thức mặc định hashCode() và equals():

- Lớp Object trong Java: tất cả các lớp trong Java đều kế thừa từ lớp Object và lớp này có các phương thức như hashCode (trả vể giá trị hash của đối tượng), getClass (trả về Class mô tả của đối tượng), toString (trả về một chuỗi biểu diễn đại diện của đối tượng)…

- Phương thức hashCode(): trả về giá trị băm (hash) của đối tượng. Giá trị băm là một giá trị số nguyên duy nhất cho một đối tượng, giúp tối ưu mục đích lưu trữ và tìm kiếm trong các cấu trúc dữ liệu dựa trên băm như HashMap, HashSet,…

- Phương thức equals(): để so sánh hai đối tượng để kiểm tra chúng có tương đương nhau hay không. Khi chưa bị ghi đè, equals so snash địac hỉ bộ nhớ của hai đối tượng để xác định sự tương đương. Tuy nhiên, ta có thể ghi đè phương thức bằng lớp con để xác định sự tương đương dựa trên các thuộc tính cụ thể của đối tượng.